

Базы данных

Учебник для высших учебных заведений

*Рекомендован Учебно-методическим объединением по образованию
в области автоматики, электроники, микроэлектроники и радиотехники
при обучении по техническим и экономическим специальностям*

Под редакцией профессора
А. Д. Хомоненко

6-е издание, дополненное

Санкт-Петербург
КОРОНА-Век
2009

ББК 32.81
УДК 004.65(075.3)
X76

Рецензенты:

Санкт-Петербургский институт информатики и автоматизации
Российской академии наук (директор института — доктор технических наук,
профессор, заслуженный деятель науки РФ *Юсупов Р. М.*)
Заместитель заведующего кафедрой Экономической информатики и АСУ СПбГУЭиФ
кандидат технических наук, доцент *Пушкина И. В.*

Хомоненко А. Д., Цыганков В. М., Мальцев М. Г.
X76 Базы данных: Учебник для высших учебных заведений / Под ред. проф. А. Д. Хомоненко. —
6-е изд., доп. — СПб.: КОРОНА-Век, 2009. — 736 с.

ISBN 978-5-7931-0527-9

Рассматриваются понятия баз данных и варианты их архитектуры. Даётся характеристика моделей представления данных, рассматриваются реляционная модель данных и проектирование реляционных баз данных, CASE-системы, защита и администрирование баз данных. Описывается технология разработки персональных баз данных и клиентской части распределенных баз данных с помощью СУБД Microsoft Access FoxPro, системы Borland C++ Builder и серверной части с помощью Microsoft SQL Server. Описываются основы публикации БД в Интернете, архитектура Web-приложений, интерфейсы CGI и ISAPI/NSAPI. Даётся общая характеристика технологии ADO .NET, Web-серверов и средств публикации баз данных в Интернете. Рассматриваются технологии публикации баз данных с использованием XML и Microsoft Access. По каждому разделу приводятся контрольные вопросы и задания.

Для студентов и слушателей высших учебных заведений, обучающихся по техническим и экономическим специальностям.

Учебник для высших учебных заведений

БАЗЫ ДАННЫХ

Верстка *Барышникова Т. К.* Дизайн обложки *Чикулаев А. А.*

ООО «КОРОНА-Век» 190005, Санкт-Петербург, Измайловский пр., д. 29

Подписано в печать 06.10.08. Формат 70×100 $\frac{1}{16}$.
Печать офсетная. Бумага газетная. Печ. л. 46. Тираж 1 000 экз. Заказ № 215

Отпечатано с готовых диапозитивов в ОАО «Техническая книга»
190005, Санкт-Петербург, Измайловский пр., 29

ISBN 978-5-7931-0527-9

© «КОРОНА-Век», 2009
© Хомоненко А. Д., Цыганков В. М.,
Мальцев М. Г., 2009

Книгу посвящаем нашим родителям.

Предисловие

Использование баз данных и информационных систем становится неотъемлемой составляющей деловой деятельности современного человека и функционирования преуспевающих организаций. В связи с этим большую актуальность приобретает освоение принципов построения и эффективного применения соответствующих технологий и программных продуктов: систем управления базами данных, CASE-систем автоматизации проектирования, средств администрирования и защиты баз данных и других.

От правильного выбора инструментальных средств создания информационных систем, определения подходящей модели данных, обоснования рациональной схемы построения базы данных, организации запросов к хранимым данным и ряда других моментов во многом зависит эффективность функционирования разрабатываемых систем. Все это требует осознанного применения теоретических положений и инструментальных средств разработки баз данных и информационных систем.

В настоящее время большой популярностью пользуется сеть Интернет. Одной из причин этого является возможность получения в реальном масштабе времени разнообразной информации, охватывающей все сферы деятельности человека. Для эффективной работы с информацией такого огромного объема нужна высокая степень ее упорядочения. Современные системы управления базами данных предоставляют развитые средства для организованного доступа к информации. Вполне логично применение технологии систем управления базами данных в сети Интернет.

Одним из основных принципов функционирования сети Интернет является представление данных в HTML-формате (на языке разметки гипертекста). Для обеспечения доступа клиентов сети Интернет к нужной информации из базы данных, находящейся на компьютере, на котором запущен Web-сервер, нужно эти данные опубликовать (представить) в HTML-формате. Публикация баз данных в Интернете осуществляется с помощью технологий, реализующих возможность размещения на Web-страницах информации из баз данных, хранящихся на Web-сервере.

Объединение Интернет-технологий и технологии систем управления базами данных (СУБД) как способ организации доступа к данным имеет следующие достоинства:

- В Интернете применяется унифицированный подход, заключающийся в использовании для доступа пользователей к хранящейся на Web-серверах информации единственной программы-обозревателя. Это позволяет стандартизовать пользовательский интерфейс.

- Использование для обмена информацией в сети платформонезависимого протокола HTTP (HyperText Transport Protocol – протокол передачи гипертекста). Обмен информацией между обозревателем и Web-сервером также осуществляется с помощью этого протокола, что позволяет стандартизовать и упростить представление данных.
- Многоуровневая архитектура сети Интернет имеет стандартные способы наращивания возможностей обозревателя и Web-сервера. Использование многоуровневой архитектуры позволяет выполнять доступ к услугам Интернета из корпоративных интрасетей и обеспечивать информационный обмен между СУБД, работающими на различных платформах (операционных системах и аппаратных средствах).
- Применение СУБД для упорядоченного хранения информации позволяет ввести стандарты и организовать на более качественном уровне хранение и извлечение данных, защиту информации, управление транзакциями с помощью языка SQL.

Результатом применения объединенной технологии Интернет/СУБД является удешевление установки и сопровождения программного обеспечения как пользователей сети Интернет, так и программного обеспечения информационных систем, построенных на основе использования СУБД.

Цель книги заключается в систематическом изложении теоретических основ построения баз данных, возможностей современных систем управления базами данных, технологий применения их для разработки и использования информационных систем, в том числе в сетях Интернет и интранет. Книга ориентирована на использование в учебном процессе, поэтому авторы стремились к доходчивому изложению материала, строгому определению понятий, четкой классификации рассматриваемых средств и технологий с указанием их достоинств и недостатков. По каждому разделу приводятся контрольные вопросы и задания,дается перечень дополнительной литературы для углубленного изучения.

Характеристика основных средств систем управления базами данных и технология разработки персональных баз данных и клиентской части распределенных баз данных с помощью Microsoft Access 2002, Borland C++ Builder 6.0, Visual FoxPro 8.0 и серверной части с помощью Microsoft SQL Server 2000 описываются применительно к современным версиям названных СУБД.

Материал четвертой части книги подготовлен таким образом, чтобы читатель имел достаточно полную информацию для создания собственного Web-приложения, публикующего базы данных в Интернете, не прибегая к дополнительной литературе. Для создания собственного Web-приложения и запуска приведенных примеров в среде операционной системы Windows 2000 Server требуется установка Web-сервера – Microsoft Internet Information Server, а для операционной системы Windows 98 нужно установить Web-сервер – Personal Web-Server. Для обеих платформ дополнительно потребуется установка драйверов – ODBC и обозревателя Internet Explorer версии не ниже

5.0. При создании собственных Web-приложений с использованием средств Java требуется пакет JDK.

Книга состоит из следующих частей.

Часть 1. Основы построения баз данных. Описываются основные понятия баз данных и информационных систем, дается общая характеристика моделей представления данных. Рассматриваются основные понятия широко распространенной реляционной модели данных, языки запросов. Описываются особенности построения и использования информационных систем в сетях, модели архитектуры клиент-сервер, управление распределенными данными, информационные системы в локальных сетях, Интернет и интранет.

Часть 2. Проектирование и использование БД. Рассматриваются вопросы проектирования реляционных баз данных (проблемы проектирования, нормализация отношений методом нормальных форм, организация связей между таблицами), метод сущность-связь, средства автоматизации проектирования (модели жизненного цикла, модели структурного и объектно-ориентированного проектирования, классификация, характеристика и рекомендации по применению CASE-систем). Описываются вопросы использования баз данных (настройка, администрирование и защита информации). Затрагиваются дополнительные вопросы применения баз данных (программно-аппаратные платформы, перспективы развития СУБД, стандартизация баз данных, современная технология ADO .NET).

Часть 3. Современные СУБД и их применение. Рассматриваются возможности и основные приемы применения современных СУБД Microsoft Access 2002, Visual FoxPro 8.0 и Microsoft SQL Server 2000, а также системы быстрой разработки приложений Borland C++ Builder 6.0, обладающей возможностями разработки приложений для работы с базами данных. Выбор названных СУБД и средств сделан ввиду их большой распространенности и высоких эксплуатационных характеристик. Рассмотрение названных программ ведется с кратким изложением ключевых вопросов. В их число включены следующие: общие сведения о системе, средства поддержки логического проектирования, средства создания основных элементов БД. К администрированию отнесены вопросы защиты информации и обслуживания БД. Материал третьей части ориентирован на эффективное решение задач проектирования, создания и администрирования БД, в том числе с применением технологии клиент-сервер. Рассматриваются особенности разработки клиентской части распределенных баз данных и организации запросов к базе данных на сервере.

Часть 4. Публикация баз данных в Интернете. Описываются основы публикации БД в Интернете, основные элементы языков HTML и XML, особенности объединения технологий Интернета и СУБД. Рассматриваются архитектура Web-приложения, объектные и программные интерфейсы ADO, OLE DB и OBDC, протоколы программирования Интернет-приложений CGI и

ISAPI/NSAPI. Дается общая характеристика Web-серверов и средств публикации баз данных в Интернете.

Рассматривается создание приложений, расширяющих возможности Web-серверов для работы с базами данных под управлением операционной системы Windows 2000. Описываются техника создания приложений и апплетов на языке Java, технология применения языка XML как стандарта организации обмена. Приводятся примеры передачи данных между XML-документами и базами данных. Рассматриваются технологии публикации баз данных с использованием Microsoft Access. Описывается применение названных систем для создания отчетов из базы данных в виде динамических HTML-страниц в популярных форматах IDC/HTX и ASP, применение языка JavaScript для создания сценариев, связывающих объекты HTML-страниц с базами данных.

При оформлении материалов использовались следующие соглашения о выделении текста: определения терминов даются полужирным начертанием; подчиненные определения — полужирным курсивом; текст, на который обращается внимание читателя, набран курсивом.

Подготовка книги основана на опыте преподавания ряда дисциплин по современным информационным технологиям, в том числе дисциплины «Базы данных», преподавателями кафедры Математического обеспечения ВКА им. А. Ф. Можайского. Отбор материала выполнен на основе общих требований программ дисциплины «Базы данных» для ряда технических и экономических специальностей высших учебных заведений, изложенных в Государственном образовательном стандарте.

Авторы благодарят руководство издательства «БХВ-Петербург» в лице В. А. Сергеева и Е. В. Кондуковой за предоставленную возможность использовать материалы из книги “Мещеряков Е. В., Хомоненко А. Д. Публикация баз данных в Интернете. — СПб.: БХВ-Петербург, 2001”, а также выражают признательность Е. В. Мещерякову за совместную подготовку материалов 4-й части книги, В. В. Гридину — за помощь в подготовке материалов 10-й главы по СУБД Access, С. В. Кирюшкину — за участие в подготовке материалов подраздела 9.4 по технологии ADO.NET, Д. В. Жарову — за помощь в подготовке материалов по серверу Apache, А. Н. Гоголеву — за помощь в отладке приложений Java, А. В. Кудашеву — за помощь в подготовке материалов по технологии применения CASE-систем для проектирования и разработки информационных систем, Д. С. Зонову — за помощь в подготовке материалов 12-й главы по СУБД Visual FoxPro.

Авторы

ОСНОВЫ ПОСТРОЕНИЯ БАЗ ДАННЫХ

1. Введение в базы данных

В разделе рассматриваются базы данных и информационные системы. Описываются основные понятия баз данных и систем управления базами данных. Даётся характеристика вариантов организации информационной системы по архитектуре клиент-сервер. Приводится классификация СУБД, и описываются основные их функции. Рассматриваются варианты создания приложений и организации взаимодействия пользователей с информационными системами.

1.1. Базы данных и информационные системы

В основе решения многих задач лежит обработка информации. Для облегчения обработки информации создаются информационные системы (ИС). Автоматизированными называют ИС, в которых применяют технические средства, в частности ЭВМ. Большинство существующих ИС являются автоматизированными, поэтому для краткости просто будем называть их ИС.

В широком понимании под определение ИС подпадает любая система обработки информации. По области применения ИС можно разделить на системы, используемые в производстве, образовании, здравоохранении, науке, военном деле, социальной сфере, торговле и других отраслях. По целевой функции ИС можно условно разделить на следующие основные категории: управляющие, информационно-справочные, поддержки принятия решений.

Заметим, что иногда используется более узкая трактовка понятия ИС как совокупности аппаратно-программных средств, задействованных для решения некоторой прикладной задачи. В организации, например, могут существовать информационные системы, на которых соответственно возложены следующие задачи: учет кадров и материально-технических средств, расчет с поставщиками и заказчиками, бухгалтерский учет и т. п.

Банк данных является разновидностью ИС, в которой реализованы функции централизованного хранения и накопления обрабатываемой информации, организованной в одну или несколько баз данных.

Банк данных (БнД) в общем случае состоит из следующих компонентов: базы (нескольких баз) данных, системы управления базами данных, словаря данных, администратора, вычислительной системы и обслуживающего персонала. Вкратце рассмотрим названные компоненты и некоторые связанные с ними важные понятия.

База данных (БД) представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

БД бывают *централизованными* (хранятся на одном компьютере) и *распределенными* (хранятся на нескольких компьютерах некоторой сети).

Логическую структуру хранимых в базе данных называют **моделью представления данных**. К основным моделям представления данных (моделям данных) относятся следующие: иерархическая, сетевая, реляционная, постреляционная, многомерная и объектно-ориентированная (см. раздел 2).

Система управления базами данных (СУБД) – это комплекс языковых и программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями. Обычно СУБД различают по используемой модели данных. Так, СУБД, основанные на использовании реляционной модели данных, называют *реляционными СУБД*.

Одними из первых СУБД являются следующие системы: IMS (IBM, 1968 г.), IDMS (Cullinet, 1971 г.), ADABAS (Software AG, 1969 г.) и ИНЭС (ВНИИСИ АН СССР, 1976 г.). Количество современных систем управления базами данных исчисляется тысячами.

Приложение представляет собой программу или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной задачи. Нами рассматриваются приложения, использующие БД. Приложения могут создаваться в среде или вне среды СУБД – с помощью системы программирования, использующей средства доступа к БД, к примеру Delphi или C++ Builder. Приложения, разработанные в среде СУБД, часто называют *приложениями СУБД*, а приложения, разработанные вне СУБД, – *внешними приложениями*.

Для работы с базой данных зачастую достаточно средств СУБД и не нужно использовать приложения, создание которых обычно требует программирования. Приложения разрабатывают главным образом в случаях, когда требуется сделать работу пользователей более удобной или автоматизировать рутинные операции с БД.

Словарь данных (СД) представляет собой подсистему БнД, предназначенную для централизованного хранения информации о структурах данных, взаимосвязях файлов БД друг с другом, типах данных и форматах их представления, принадлежности данных пользователям, кодах защиты и разграничения доступа и т. п.

Словарь данных, иначе называемый системным каталогом, как следует из определения, является хранилищем служебной информации о данных в базе («данных о данных», или метаданных).

Функционально СД присутствует во всех БнД, но не всегда выполняющий эти функции компонент имеет именно такое название. Чаще всего функции СД выполняются СУБД и вызываются из основного меню системы или реализуются с помощью ее утилит.

Если СД является частью БД, то его называют *интегрированным СД*, в противном случае СД является *автономным*. Автономные словари данных обычно используют не только в интересах собственно данных базы, но и в целях управления другими информационными ресурсами организаций при разработке структур баз данных на этапе проектирования, для ведения документации, управления проектами и т. д.

Стандартизация интерфейса СД привела к разработке службы словаря информационных ресурсов (Information Resource Dictionary System – IRDS). Служба IRDS имеет четыре интерфейса: графический, командный язык, экспорта/импорта и прикладных программ. Реализация IRDS представляет собой программный инструмент для унифицированного управления различными информационными ресурсами организации группами пользователей и приложениями. Введение IRDS может быть целесообразно на ранних этапах проектирования БД организации, когда необходимо отложить привязку БД к конкретной СУБД. Кроме того, с помощью служб IRDS можно переносить информацию между IRDS-совместимыми СД различных СУБД (независимо от используемой в них модели данных).

Администратор базы данных (АБД) есть лицо или группа лиц, отвечающих за выработку требований к БД, ее проектирование, создание, эффективное использование и сопровождение. В процессе эксплуатации АБД обычно следит за функционированием информационной системы, обеспечивает защиту от несанкционированного доступа, контролирует избыточность, непротиворечивость, сохранность и достоверность хранимой в БД информации. Для однопользовательских информационных систем функции АБД обычно возлагаются на лиц, непосредственно работающих с приложением БД.

В вычислительной сети АБД, как правило, взаимодействует с *администратором сети*. В обязанности последнего входят контроль за функционированием аппаратно-программных средств сети, реконфигурация сети, восстановление программного обеспечения после сбоев и отказов оборудования, профилактические мероприятия и обеспечение разграничения доступа.

Вычислительная система (ВС) представляет собой совокупность взаимосвязанных и согласованно действующих ЭВМ или процессоров и других устройств, обеспечивающих автоматизацию процессов приема, обработки и выдачи информации потребителям. Поскольку основными функциями БнД

являются хранение и обработка данных, то используемая ВС, наряду с приемлемой мощностью центральных процессоров (ЦП) должна иметь достаточный объем оперативной и внешней памяти прямого доступа.

Обслуживающий персонал выполняет функции поддержания технических и программных средств в работоспособном состоянии. Он проводит профилактические, регламентные, восстановительные и другие работы по планам, а также по мере необходимости.

1.2. Архитектура информационной системы

Эффективность функционирования информационной системы (ИС) во многом зависит от ее архитектуры. В настоящее время перспективной является архитектура клиент-сервер. В достаточно распространенном варианте она предполагает наличие компьютерной сети и распределенной базы данных, включающей корпоративную базу данных (КБД) и персональные базы данных (ПБД). КБД размещается на компьютере-сервере, ПБД размещаются на компьютерах сотрудников подразделений, являющихся клиентами корпоративной БД.

Сервером определенного ресурса в компьютерной сети называется компьютер (программа), управляющий этим ресурсом, **клиентом** – компьютер (программа), использующий этот ресурс. В качестве ресурса компьютерной сети могут выступать, к примеру, базы данных, файловые системы, службы печати, почтовые службы. Тип сервера определяется видом ресурса, которым он управляет. Например, если управляемым ресурсом является база данных, то соответствующий сервер называется *сервером базы данных*.

Достоинством организации информационной системы по архитектуре клиент-сервер является удачное сочетание *централизованного хранения*, обслуживания и коллективного доступа к общей корпоративной информации с индивидуальной работой пользователей над персональной информацией. Архитектура клиент-сервер допускает различные варианты реализации.

Исторически первыми появились распределенные ИС с применением **файл-сервера** (рис. 1.1). В таких ИС по запросам пользователей файлы базы данных передаются на персональные компьютеры (ПК), где и производится их обработка. **Недостатком** такого варианта архитектуры является высокая интенсивность передачи обрабатываемых данных. Причем зачастую передаются избыточные данные: вне зависимости от того сколько записей из базы данных требуется пользователю, файлы базы данных передаются целиком.

Структура распределенной ИС, построенной по архитектуре клиент-сервер с использованием сервера баз данных, показана на рис. 1.2. При такой архитектуре сервер базы данных обеспечивает выполнение основного объема обработки данных. Формируемые пользователем или приложением запросы поступают

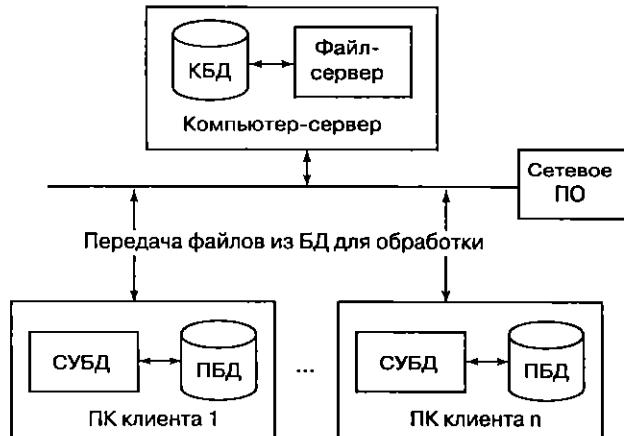


Рис. 1.1. Структура ИС с файл-сервером

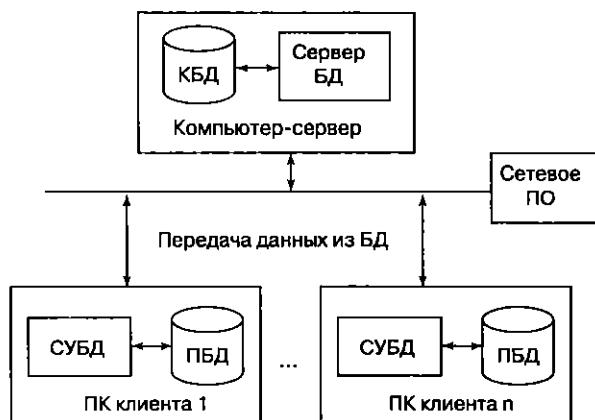


Рис. 1.2. Структура ИС с сервером баз данных

ют к серверу БД в виде инструкций языка SQL. Сервер базы данных выполняет поиск и извлечение нужных данных, которые затем передаются на компьютер пользователя. *Достоинством* такого подхода в сравнении предыдущим является заметно меньший объем передаваемых данных.

Основные варианты построения распределенных БД по архитектуре клиент-сервер рассматриваются в разделе 4.

Для создания и управления персональными БД и приложений, работающих с ними, используются СУБД, такие как Access и Visual FoxPro фирмы Microsoft, Paradox фирмы Borland.

Корпоративная БД создается, поддерживается и функционирует под управлением сервера БД, например Microsoft SQL Server или Oracle Server.

В зависимости от размеров организации и особенностей решаемых задач информационная система может иметь одну из следующих конфигураций:

- компьютер-сервер, содержащий корпоративную и персональные базы;
- компьютер-сервер и персональные компьютеры с ПБД;
- несколько компьютеров-серверов и персональных компьютеров с ПБД.

Использование архитектуры клиент-сервер дает возможность постепенного наращивания информационной системы предприятия, во-первых, по мере развития предприятия; во-вторых, по мере развития самой информационной системы.

Разделение общей БД на корпоративную БД и персональные БД позволяет уменьшить сложность проектирования БД по сравнению с централизованным вариантом, а значит, снизить вероятность ошибок при проектировании и стоимость проектирования.

Важнейшим достоинством применения БД в информационных системах является обеспечение независимости данных от прикладных программ. Это дает возможность пользователям не заниматься проблемами представления данных на физическом уровне: размещения данных в памяти, методов доступа к ним и т. д.

Такая независимость достигается поддерживаемым СУБД многоуровневым представлением данных в БД на логическом (пользовательском) и физическом уровнях. Благодаря СУБД и наличию логического уровня представления данных обеспечивается отделение концептуальной (понятийной) модели БД от ее физического представления в памяти ЭВМ.

1.3. Системы управления базами данных

В этом подразделе приводится классификация СУБД и рассматриваются основные их функции. В качестве основных классификационных признаков можно использовать следующие: вид программы, характер использования, модель данных. Названные признаки существенно влияют на целевой выбор СУБД и эффективность использования разрабатываемой информационной системы.

Классификация СУБД. В общем случае под СУБД можно понимать любой программный продукт, поддерживающий процессы создания, ведения и использования БД. Рассмотрим, какие из имеющихся на рынке программ имеют отношение к БД и в какой мере они связаны с базами данных.

К СУБД относятся следующие основные виды программ:

- полнофункциональные СУБД;
- серверы БД;

- клиенты БД;
- средства разработки программ работы с БД.

Полнофункциональные СУБД (ПФСУБД) представляют собой традиционные СУБД, которые сначала появились для больших машин, затем для мини-машин и для ПЭВМ. Из числа всех СУБД современные ПФСУБД являются наиболее многочисленными и мощными по своим возможностям. К ПФСУБД относятся, например, такие пакеты, как Clarion Database Developer, DataEase, DataFlex, dBase IV, Microsoft Access, Microsoft FoxPro и Paradox R:BASE.

Обычно ПФСУБД имеют развитый интерфейс, позволяющий с помощью команд меню выполнять основные действия с БД: создавать и модифицировать структуры таблиц, вводить данные, формировать запросы, разрабатывать отчеты, выводить их на печать и т. п. Для создания запросов и отчетов не обязательно программирование, а удобно пользоваться языком QBE (Query By Example — формулировки запросов по образцу, см. подраздел 3.8). Многие ПФСУБД включают средства программирования для профессиональных разработчиков.

Некоторые системы имеют в качестве вспомогательных и дополнительные средства проектирования схем БД или CASE-подсистемы. Для обеспечения доступа к другим БД или к данным SQL-серверов полнофункциональные СУБД имеют факультативные модули.

Серверы БД предназначены для организации центров обработки данных в сетях ЭВМ. Эта группа БД в настоящее время менее многочислена, но их количество постепенно растет. Серверы БД реализуют функции управления базами данных, запрашиваемые другими (клиентскими) программами обычно с помощью операторов SQL.

Примерами серверов БД являются следующие программы: NetWare SQL (Novell), MS SQL Server (Microsoft), InterBase (Borland), SQLBase Server (Gupta), Intelligent Database (Ingress).

В роли **клиентских программ** для серверов БД в общем случае могут использоваться различные программы: ПФСУБД, электронные таблицы, текстовые процессоры, программы электронной почты и т. д. При этом элементы пары «клиент — сервер» могут принадлежать одному или разным производителям программного обеспечения.

В случае, когда клиентская и серверная части выполнены одной фирмой, естественно ожидать, что распределение функций между ними выполнено рационально. В остальных случаях обычно преследуется цель обеспечения доступа к данным «любой ценой». Примером такого соединения является случай, когда одна из полнофункциональных СУБД играет роль сервера, а вторая СУБД (другого производителя) — роль клиента. Так, для сервера БД SQL Server (Microsoft) в роли клиентских (фронтальных) программ могут выступать многие СУБД, такие как dBASE IV, Blyth Software, Paradox, DataEase, Focus, 1-2-3, MDBS III, Revelation и другие.

Средства разработки программ работы с БД могут использоваться для создания разновидностей следующих программ:

- клиентских программ;
- серверов БД и их отдельных компонентов;
- пользовательских приложений.

Программы первого и второго вида довольно малочисленны, так как предназначены, главным образом, для системных программистов. Пакетов третьего вида гораздо больше, но меньше, чем полнофункциональных СУБД.

К средствам разработки пользовательских приложений относятся системы программирования, например Clipper, разнообразные библиотеки программ для различных языков программирования, а также пакеты автоматизации разработок (в том числе систем типа клиент-сервер). В числе наиболее распространенных можно назвать следующие инструментальные системы: Delphi и Power Builder (Borland), Visual Studio (Microsoft), SILVERRUN (Computer Advisers Inc.), S-Designor (SDP и Powersoft) и ERwin (LogicWorks).

Если говорить о конкретных системах программирования (для языков C++, C#, Visual Basic, Java и др.), то все они содержат некоторые средства доступа к наиболее широко используемым БД.

Кроме перечисленных средств, для управления данными и организации обслуживания БД используются различные дополнительные средства, к примеру мониторы транзакций (см. подраздел 4.2).

По характеру использования СУБД делят на *персональные и многопользовательские*.

Персональные СУБД обычно обеспечивают возможность создания персональных БД и недорогих приложений, работающих с ними. Персональные СУБД или разработанные с их помощью приложения зачастую могут выступать в роли клиентской части многопользовательской СУБД. К персональным СУБД, например, относятся Visual FoxPro, Paradox, Clipper, dBase, Access и др.

Многопользовательские СУБД включают в себя сервер БД и клиентскую часть и, как правило, могут работать в неоднородной вычислительной среде (с разными типами ЭВМ и операционными системами). К многопользовательским СУБД относятся, например, СУБД Oracle и Informix.

В зависимости от способа хранения и обработки БД (централизованного или децентрализованного) СУБД можно разделить на два класса: *централизованные* (или обычные) СУБД и *децентрализованные* (или распределенные) СУБД. В обычных СУБД данные хранятся в том же месте, где и программы их управления. В распределенных СУБД как программное обеспечение, так и данные распределены по узлам сети. Распределенные СУБД могут быть *однородными* или *неоднородными*. Неоднородность СУБД может проявляться в отличии поддерживаемых моделей данных, типов данных, языков запросов, фирм-разработчиков и т. д.

Одной из разновидностей распределенных СУБД являются *мультибазовые системы*, в которых управление каждым из узлов осуществляется автономно. В мультибазовых СУБД производится такая интеграция локальных систем, при которой не требуется изменение существующих СУБД и в то же время конечным пользователям предоставляется доступ к совместно используемым данным. Пользователи локальных СУБД получают возможность управлять данными собственных узлов без централизованного контроля, который присутствует в обычных распределенных СУБД. Примером мультибазовой СУБД является система UniSQL компании Cincom Corporation.

В зависимости от возможности распараллеливания процесса обработки данных выделяют *СУБД с последовательной и параллельной обработкой (параллельные СУБД)*. Параллельные СУБД функционируют в многопроцессорной вычислительной системе (как правило, со множеством устройств хранения данных) или в сети компьютеров.

По используемой модели данных СУБД (как и БД), разделяют на иерархические, сетевые, реляционные, объектно-ориентированные и другие типы. Некоторые СУБД могут одновременно поддерживать несколько моделей данных.

С точки зрения пользователя, СУБД реализует *функции хранения, изменения (пополнения, редактирования и удаления) и обработки информации, а также разработки и получения различных выходных документов*.

Для работы с хранящейся в базе данных информацией СУБД предоставляет программам и пользователям следующие два типа *языков*:

- язык описания данных — высокоуровневый непрограммный язык декларативного типа, предназначенный для описания логической структуры данных;
- язык манипулирования данными — совокупность конструкций, обеспечивающих выполнение основных операций по работе с данными: ввод, модификацию и выборку данных по запросам.

Названные языки в различных СУБД могут иметь отличия. Наибольшее распространение получили два стандартизованных языка: QBE (Query By Example) — язык запросов по образцу и SQL (Structured Query Language) — структурированный язык запросов. QBE в основном обладает свойствами языка *манипулирования данными*, SQL сочетает в себе свойства языков обоих типов — *описания и манипулирования данными*.

Перечисленные выше функции СУБД, в свою очередь, используют следующие основные функции более низкого уровня, которые назовем *низкоуровневыми*:

- управление данными во внешней памяти;
- управление буферами оперативной памяти;
- управление транзакциями;
- ведение журнала изменений в БД;
- обеспечение целостности и безопасности БД.

• КИПИДАЛ
361540-Э011

Дадим краткую характеристику необходимости и особенностям реализации перечисленных функций в современных СУБД.

Реализация функции *управления данными во внешней памяти* в разных системах может различаться и на уровне управления ресурсами (используя файловые системы ОС или непосредственное управление устройствами ПЭВМ), и по логике самих алгоритмов управления данными. В основном методы и алгоритмы управления данными являются «внутренним делом» СУБД и прямого отношения к пользователю не имеют. Качество реализации этой функции наиболее сильно влияет на эффективность работы специфических ИС, например, с огромными БД, со сложными запросами, большим объемом обработки данных.

Необходимость буферизации данных и как следствие реализации функции *управления буферами* оперативной памяти обусловлено тем, что объем оперативной памяти меньше объема внешней памяти.

Буферы представляют собой области оперативной памяти, предназначенные для ускорения обмена между внешней и оперативной памятью. В буферах временно хранятся фрагменты БД, данные из которых предполагается использовать при обращении к СУБД или планируется записать в базу после обработки.

Механизм транзакций используется в СУБД для поддержания целостности данных в базе. **Транзакцией** называется некоторая неделимая последовательность операций над данными БД, которая отслеживается СУБД от начала и до завершения. Если по каким-либо причинам (сбои и отказы оборудования, ошибки в программном обеспечении, включая приложение) транзакция остается незавершенной, то она отменяется.

В зависимости от времени, требуемого для выполнения, выделяют *обычные и продолжительные транзакции*. Продолжительные транзакции могут охватывать часы, дни и даже месяцы. Такие транзакции могут возникать в процессе проектирования и разработки сложных систем крупным коллективом людей. Кроме того, помимо обычных *плоских транзакций*, используется модель *вложенных транзакций*. В последнем случае транзакция рассматривается как набор взаимосвязанных подзадач (субтранзакций), каждая из которых также может состоять из произвольного количества субтранзакций.

Говорят, что транзакции присущи три основных свойства:

- атомарность (выполняются все входящие в транзакцию операции или ни одна);
- серализумость (отсутствует взаимное влияние выполняемых в одно и то же время транзакций);
- долговечность (даже крах системы не приводит к утрате результатов зафиксированной транзакции).

Примером транзакции является операция перевода денег с одного счета на другой в банковской системе. Здесь необходим, по крайней мере, двухшаговый

процесс. Сначала снимают деньги с одного счета, затем добавляют их к другому счету. Если хотя бы одно из действий не выполнится успешно, результат операции окажется неверным и будет нарушен баланс между счетами.

Контроль транзакций важен в однопользовательских и в многопользовательских СУБД, где транзакции могут быть запущены параллельно. В последнем случае говорят о сериализации транзакций. Под *сериализацией* параллельно выполняемых транзакций понимается составление такого плана их выполнения (сериального плана), при котором суммарный эффект реализации транзакций эквивалентен эффекту их последовательного выполнения.

При параллельном выполнении смеси транзакций возможно возникновение конфликтов (блокировок), разрешение которых является функцией СУБД. При обнаружении таких случаев обычно производится «откат» путем отмены изменений, произведенных одной или несколькими транзакциями.

Ведение журнала изменений в БД (журнализация изменений) выполняется СУБД для обеспечения надежности хранения данных в базе при наличии аппаратных сбоев и отказов, а также ошибок в программном обеспечении.

Журнал СУБД – это особая БД или часть основной БД, непосредственно недоступная пользователю и используемая для записи информации обо всех изменениях базы данных. В различных СУБД в журнал могут заноситься записи, соответствующие изменениям в СУБД на разных уровнях: от минимальной внутренней операции модификации страницы внешней памяти до логической операции модификации БД (например, вставки записи, удаления столбца, изменения значения в поле) и даже транзакции.

Для эффективной реализации функции ведения журнала изменений в БД необходимо обеспечить повышенную надежность хранения и поддержания в рабочем состоянии самого журнала. Иногда для этого в системе хранят несколько копий журнала.

Обеспечение целостности БД составляет необходимое условие успешного функционирования БД, особенно для случая использования БД в сетях. **Целостность БД** есть свойство базы данных, означающее, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область информация. Поддержание целостности БД включает проверку целостности и ее восстановление в случае обнаружения противоречий в базе данных. Целостное состояние БД описывается с помощью *ограничений целостности* в виде условий, которым должны удовлетворять хранимые в базе данные. Примером таких условий может служить ограничение диапазонов возможных значений атрибутов объектов, сведения о которых хранятся в БД, или отсутствие повторяющихся записей в таблицах реляционных БД.

Обеспечение безопасности достигается в СУБД шифрованием прикладных программ, данных, защиты паролем, поддержкой уровней доступа к базе данных и к отдельным ее элементам (таблицам, формам, отчетам и т. д.).

1.4. Локальные информационные системы

Функциональные части информационной системы могут размещаться на одном или на нескольких компьютерах. Рассмотрим варианты организации ИС на одном ПК. Соответствующую ИС обычно называют локальной или однопользовательской (хотя последнее не совсем строго, поскольку на одном компьютере поочередно могут работать несколько пользователей). Более сложные варианты организации ИС рассматриваются в разделе 4.

Организация функционирования локальной ИС на одном компьютере в среде некоторой операционной системы (ОС) возможна с помощью следующих вариантов использования программных средств:

- «полной» СУБД;
- приложения и «усеченной» (ядра) СУБД;
- независимого приложения.

Первый способ обычно применяется в случаях, когда в дисковой памяти компьютера помещается вся СУБД и она часто используется для доработки приложения (рис. 1.3).

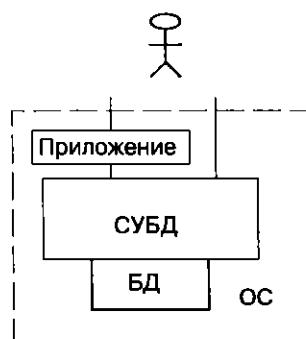


Рис. 1.3. Использование приложения и СУБД

Взаимодействие пользователя с СУБД происходит напрямую через пользовательский (терминальный) интерфейс СУБД, либо с помощью приложения. Приложение выполняется в режиме *интерпретации* (см. подраздел 1.5).

Основное достоинство схемы — простота разработки и сопровождения БД и приложений при наличии развитых соответствующих средств разработки и сервисных средств. Недостатком этой схемы являются затраты дисковой памяти на хранение программы СУБД и оперативной памяти для исполняемого кода.

Приложение с ядром СУБД (рис. 1.4) используют для достижения следующих целей:

- уменьшения объема занимаемого СУБД пространства жесткого диска и оперативной памяти;
- повышения скорости работы приложения;
- защиты приложения от модификации со стороны пользователя (обычно ядро не содержит средств разработки приложений).

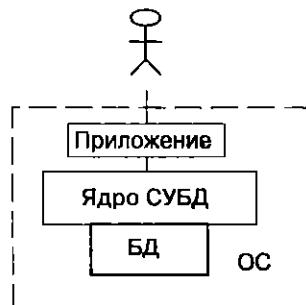


Рис. 1.4. Использование приложения и ядра СУБД

Примером такого подхода является использование модуля FoxRun системы FoxBase+. Из современных СУБД отметим Microsoft Access, включающую дополнительный пакет Microsoft Access Developer's Toolkit. С его помощью можно создавать переносимую на дискетах «укороченную» (теп-time) версию Microsoft Access, не содержащую инструментов разработки.

Достоинствами использования ядра СУБД по сравнению с использованием полной версии СУБД являются: меньшее потребление ресурсов памяти компьютера, ускорение работы приложения и возможность защиты приложения от модификации. К основным *недостаткам* можно отнести все еще значительный объем дисковой памяти, необходимой для хранения ядра СУБД, и недостаточно высокое быстродействие работы приложений (выполнение приложения по-прежнему происходит путем *интерпретации*).

При третьем способе организации ИС исходная программа предварительно *компилируется* — преобразуется в последовательность исполняемых машинных команд. В результате получается готовая к выполнению независимая программа, не требующая для своей работы ни всей СУБД, ни ее ядра (рис. 1.5). Заметим, что с точки зрения выполнения основных функций хранения и обработки данных такая программа мало отличается от приложения, работающего под управлением СУБД или ее ядра.

Основными достоинствами этого варианта по сравнению с двумя предыдущими являются: экономия внешней и оперативной памяти компьютера, ус-

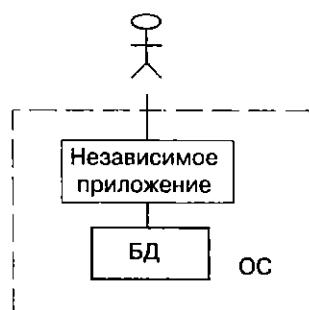


Рис. 1.5. Использование независимого приложения

корение выполнения приложения и полная защита приложения от модификации (случай дизассемблирования и вставки своего кода и ему подобные в расчет не берутся). К недостаткам можно отнести трудоемкость доработки приложений и отсутствие возможности использовать стандартные средства СУБД по обслуживанию БД.

1.5. Способы разработки и выполнения приложений

Современные СУБД позволяют решать широкий круг задач по работе с базами данных без разработки приложения. Тем не менее есть случаи, когда целесообразно разработать приложение. Например, если требуется автоматизация манипуляций с данными, терминальный интерфейс СУБД недостаточно развит, либо имеющиеся в СУБД стандартные функции по обработке информации не устраивают пользователя. Для разработки приложений СУБД должна иметь программный интерфейс, основу которого составляют функции и/или процедуры соответствующего языка программирования.

Существующие СУБД поддерживают следующие технологии (и их комбинации) разработки приложений:

- ручное кодирование программ (Clipper, FoxPro, Paradox);
- создание текстов приложений с помощью генераторов (FoxApp в FoxPro, Personal Programmer в Paradox);
- автоматическая генерация готового приложения методами визуального программирования (Delphi, Access, Paradox for Windows).

При *ручном кодировании* программисты вручную набирают текст программ приложений, после чего выполняют их отладку.

Использование *генераторов* упрощает разработку приложений, поскольку при этом можно получать программный код без ручного набора. Генераторы приложений облегчают разработку основных элементов приложений

(меню, экранах форм, запросов и т. д.), но зачастую не могут полностью исключить ручное кодирование.

Средства *визуального программирования* приложений являются дальнейшим развитием идеи использования генераторов приложений. Приложение при этом строится из готовых «строительных блоков» с помощью удобной интегрированной среды. При необходимости разработчик легко может вставить в приложение свой код. Интегрированная среда, как правило, предоставляет мощные средства создания, отладки и модификации приложений. Использование средств визуального программирования позволяет в кратчайшие сроки создавать более надежные, привлекательные и эффективные приложения по сравнению с приложениями, полученными первыми двумя способами.

Разработанное приложение обычно состоит из одного или нескольких файлов операционной системы.

Если основным файлом приложения является исполняемый файл (например, *exe-файл*), то это приложение, скорее всего, является *независимым приложением*, которое выполняется автономно от среды СУБД. Получение независимого приложения на практике осуществляется путем *компиляции* исходных текстов программ, полученных различными способами: путем набора текста вручную, а также полученных с помощью генератора приложения или среды визуального программирования.

Независимые приложения позволяют получать, например, СУБД FoxPro и систему визуального программирования Delphi. Отметим, что с помощью средств Delphi обычно независимые приложения не разрабатывают, так как это достаточно трудоемкий процесс, а привлекают процессор баз данных BDE (Borland DataBase Engine), играющий роль ядра СУБД. Одним из первых средств разработки приложений для персональных ЭВМ является система Clipper, представляющая собой «чистый компилятор».

Во многих случаях приложение не может исполняться без среды СУБД. Выполнение приложения состоит в том, что СУБД анализирует содержимое файлов приложения (в частном случае — это текст исходной программы) и автоматически строит необходимые исполняемые машинные команды. Другими словами, приложение выполняется методом *интерпретации*.

Режим интерпретации реализован во многих современных СУБД, например Access, Visual FoxPro и Paradox, а также в СУБД недавнего прошлого, к примеру FoxBase и FoxPro.

Кроме этого, существуют системы, использующие промежуточный вариант между компиляцией и интерпретацией — так называемую *псевдокомпиляцию*. В таких системах исходная программа путем компиляции преобразуется в промежуточный код (псевдокод) и записывается на диск. В этом виде ее в некоторых системах разрешается даже редактировать, но главная цель псевдокомпиляции — преобразовать программу к виду, ускоряющему процесс ее интерпретации. Такой прием широко применялся в

СУБД, работающих под управлением DOS, например Foxbase+ и Paradox 4.0/4.5 for DOS.

В СУБД, работающих под управлением Windows, псевдокод чаще используют для того, чтобы запретить модифицировать приложение. Это полезно для защиты от случайной или преднамеренной порчи работающей программы. Например, такой прием применен в СУБД Paradox for Windows, где допускается разработанные экранные формы и отчеты преобразовывать в соответствующие объекты, не поддающиеся редактированию.

Некоторые СУБД предоставляют пользователю возможность выбора варианта разработки приложения: как интерпретируемого СУБД программного кода или как независимой программы.

Достоинством применения *независимых приложений* является то, что время выполнения машинной программы обычно меньше, чем при интерпретации. Такие приложения целесообразно использовать на слабых машинах и в случае установки систем «под ключ», когда необходимо закрыть приложение от доработок со стороны пользователей.

Важным *достоинством* применения *интерпретируемых* приложений является легкость их модификации. Если готовая программа подвергается частым изменениям, то для их внесения нужна инструментальная система, то есть СУБД или аналогичная среда. Для интерпретируемых приложений такой инструмент всегда под рукой, что очень удобно.

Другим серьезным *достоинством* систем с интерпретацией является то, что хорошие СУБД обычно имеют мощные сервисные средства (контроль целостности данных, защита от несанкционированного доступа, динамическая оптимизация выполнения запросов, архивация данных и прочее). В последних упомянутые функции приходится программировать вручную либо оставлять на совести администраторов.

При выборе средств для разработки приложения следует учитывать три основных фактора: ресурсы компьютера, особенности приложения (потребность в модификации функций программы, время на разработку, необходимость использования сервисных функций) и цель разработки (отчуждаемый программный продукт или система автоматизации своей повседневной деятельности).

Для пользователя, имеющего современный компьютер и планирующего создать несложное приложение, по всей видимости, больше подойдет СУБД интерпретирующего типа. Напомним, что такие системы достаточно мощны, имеют высоконивневые средства, удобны для разработки и отладки, позволяют быстро выполнить разработку и обеспечивают удобное сопровождение и модификацию приложения.

При использовании компьютера со слабыми характеристиками лучше остановить свой выбор на системе со средствами разработки независимых приложений. При этом следует иметь в виду, что малейшее изменение в приложении влечет за собой циклическое повторение этапов программирования,

компиляции и отладки программы. Разница в выполнении независимого приложения и выполнения приложения в режиме интерпретации колеблется в пределах миллисекунд в пользу независимого приложения. В то же время разница во времени подготовки приложения к его использованию обычно составляет величины порядка минуты—часы в пользу систем с интерпретацией.

1.6. Схема обмена данными при работе с БД

Пользователю любой категории (администратору БД, разработчику приложения, обычному пользователю) для грамотного решения задач полезно представлять вычислительный процесс, происходящий в ОС при работе с БД. Раскроем внутренние механизмы этого процесса на примере наиболее общего случая организации ИС, функционирующей на одном ПК, — когда пользователь работает с «полной» версией программы СУБД (рис. 1.3). Варианты, представленные на рис. 1.4 и рис. 1.5, можно считать частными случаями.

При работе пользователя с базой данных над ее содержимым выполняются следующие основные операции: выбор, добавление, модификация (замена) и удаление данных. Рассмотрим, как происходит обмен данными между отдельным пользователем и персональной СУБД при выполнении наиболее часто используемой операции *выбора данных*. Обмен данными между пользователем и БД для других операций отличается несущественно.

Схематично обмен данными при работе пользователя с БД можно представить так, как показано на рис. 1.6, где обычными стрелками обозначены связи по управлению, утолщенные — связи по информации.

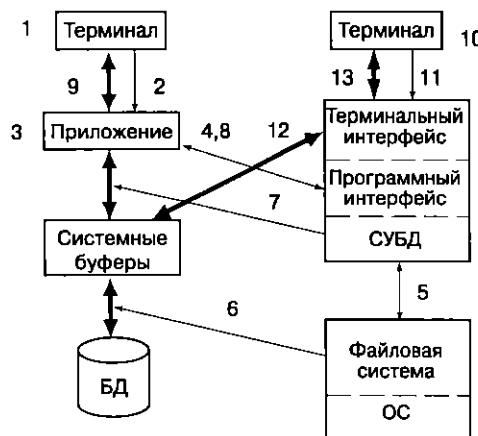


Рис. 1.6. Схема обмена данными при работе с БД

Цикл взаимодействия пользователя с БД с помощью приложения можно разделить на следующие основные этапы:

1. Пользователь терминала (1) в процессе диалога с приложением формулирует запрос (2) на некоторые данные из БД.
2. Приложение (3) на программном уровне средствами языка манипулирования данными формулирует запрос (4), с которым обращается к СУБД.
3. Используя свои системные управляющие блоки и таблицы, СУБД с помощью *словаря данных* определяет местоположение требуемых данных и обращается (5) за ними к ОС.
4. Программы методов доступа файловой системы ОС считывают (6) из внешней памяти искомые данные и помещают их в системные буферы СУБД.
5. Преобразуя полученные данные к требуемому формату, СУБД пересыпает их (7) в соответствующую область программы и сигнализирует (8) о завершении операции каким-либо образом (например, кодом возврата).
6. Результаты выбора данных из базы приложение (3) отображает (9) на терминале пользователя (1).

В случае работы пользователя в диалоговом режиме с СУБД (без приложения) цикл взаимодействия пользователя с БД упрощается. Его можно представить следующими этапами:

1. Пользователь терминала (10) формулирует на языке запросов СУБД, например QBE, по связи (11) требование на выборку некоторых данных из базы.
2. СУБД определяет местоположение требуемых данных и обращается (5) за ними к ОС, которая считывает (6) из внешней памяти искомые данные и помещает их в системные буферы СУБД.
3. Информация из системных буферов преобразуется (12) к требуемому формату, после чего отображается (13) на терминале пользователя (10).

Напомним, что описанная схема поясняет, как функционирует СУБД с одним пользователем на отдельной ПЭВМ.

Если компьютер и ОС поддерживают многопользовательский режим работы, то в такой вычислительной системе может функционировать *многопользовательская СУБД*. Последняя, в общем случае, позволяет одновременно обслуживать нескольких пользователей, работающих непосредственно с СУБД или с приложениями (каждое из которых может поддерживать работу с одним или несколькими пользователями).

Иногда к вычислительной системе подключается так называемый «удаленный пользователь», находящийся на некотором удалении от ЭВМ и соединенный с ней при помощи какой-либо передающей среды (интерфейс ЭВМ, телефонный канал связи, радиоканал, оптико-волоконная линия и т. д.). Чаще всего такой пользователь программным способом эмулируется под обычного локального пользователя. СУБД, как правило, этой подмены «не замечает» и работает по обслуживанию запросов обычным образом.

В многопользовательских СУБД при выполнении различных операций параллельно происходят процессы, подобные описанным выше и показанным на рис. 1.6.

При обслуживании нескольких параллельных источников запросов (от пользователей и приложений) СУБД так планирует использование своих ресурсов и ресурсов ЭВМ, чтобы обеспечить независимое или почти независимое выполнение операций, порождаемых запросами.

Многопользовательские СУБД часто применяются на больших и средних ЭВМ, где основным режимом использования ресурсов является коллективный доступ.

На персональных ЭВМ пользователь обычно работает один, но с различными программами, в том числе и одновременно (точнее, попеременно). Иногда такими программами оказываются СУБД: различные программы или разные копии одной и той же СУБД. Последняя ситуация возникает, например, при работе с различными базами данных с помощью СУБД Access.

Технология одновременной работы пользователя с несколькими программами неплохо реализована в Windows. Здесь каждая выполняемая программа имеет свое окно взаимодействия с пользователем и имеются удобные средства переключения между программами. При работе в Windows СУБД избавлена от необходимости поддержания нескольких сеансов работы с пользователями.

Контрольные вопросы и задания

1. Дайте определение понятия информационной системы в широком и узком смысле.
2. Что представляет собой банк данных и какие компоненты входят в его состав?
3. Каково назначение СУБД?
4. Назовите основные модели данных.
5. Дайте определение приложения, укажите, в каких случаях оно разрабатывается.
6. Укажите назначение словаря данных.
7. Перечислите функции администратора базы данных.
8. Охарактеризуйте архитектуру клиент-сервер и назовите варианты ее реализации, укажите достоинства и недостатки.
9. Изобразите структуру информационной системы с файл-сервером.
10. Изобразите структуру информационной системы с сервером баз данных.
11. Дайте классификацию СУБД.
12. Назовите основные функции СУБД.
13. Укажите понятие транзакции. Назовите виды транзакций.

14. Назовите основные способы работы пользователя с базой данных при решении прикладных задач.
15. Укажите технологии создания приложений работы с базами данных.
16. Охарактеризуйте способы выполнения приложений работы с базами данных.
17. Изобразите схему обмена данными пользователя с БД для следующих операций обработки данных:
1) выборки; 2) добавления; 3) модификации; 4) удаления.
18. Дайте характеристику многопользовательским СУБД.

Литература

1. *Бородаев В. А., Кустов В. Н. Банки и базы данных: Учебное пособие.* Л.: ВИКИ, 1989.
2. Вычислительные системы и их программное обеспечение: модели, методы и средства исследования / Под ред. профессоров Ю. И. Рыжикова и А. Д. Хомоненко. Учебник для вузов. Министерство обороны РФ, 1995.
3. *Конноли Т., Бэгг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика.* / Пер. с англ. – 3-е изд. – М.: Издательский дом «Вильямс», 2003. – 1440 с.
4. *Кузнецов С. Д. Введение в СУБД. Часть 2 // Системы Управления Базами Данных, № 2, 1995.* С. 116–124.
5. Основы современных компьютерных технологий: Учебник / Под ред. проф. Хомоненко А. Д. Авторы: Брякалов Г. А., Войцеховский С. В., Воробьев Е. Г., Гофман В. Э., Гридин В. В., Дрюков Ю. П., Замула А. А., Захаров А. И., Компаниец Р. И., Липецких А. Г., Рыжиков Ю. И., Хомоненко А. Д., Цыганков В. М. – СПб: КОРОНА прнт, 2005. – 672 с.
6. Системы управления базами данных и знаний: Справ. изд. / Наумов А. Н., Вендрев А. М., Иванов В. К. и др.; Под ред. А. Н. Наумова. М.: Финансы и статистика, 1991.
7. *Советов Б. Я., Цехановский В. В., Чертовский В. Д. Базы данных. Теория и практика.* – М.: Высшая школа, 2005.
8. *Четвериков В. Н. и др. Базы и банки данных: Учебник для вузов по спец. «АСУ» / Под ред. В. Н. Четверикова.* М.: Вышш. шк., 1987.

2. Модели и типы данных

Хранимые в базе данные имеют определенную логическую структуру — иными словами, описываются некоторой **моделью представления данных** (моделью данных), поддерживаемой СУБД. К числу классических относятся следующие модели данных:

- иерархическая,
- сетевая,
- реляционная.

Кроме того, в последние годы появились и стали более активно внедряться на практике следующие модели данных:

- постреляционная,
- многомерная,
- объектно-ориентированная.

Разрабатываются также всевозможные системы, основанные на других моделях данных, расширяющих известные модели. В их числе можно назвать объектно-реляционные, дедуктивно-объектно-ориентированные, семантические, концептуальные и ориентированные модели. Некоторые из этих моделей служат для интеграции баз данных, баз знаний и языков программирования.

В некоторых СУБД поддерживаются одновременно несколько моделей данных. Например, в системе ИНТЕРБАЗА для приложений применяется сетевой язык манипулирования данными, а в пользовательском интерфейсе реализованы языки SQL и QBE.

2.1. Иерархическая модель

В иерархической модели связи между данными можно описать с помощью упорядоченного графа (или дерева). Упрощенное представление связей между данными в иерархической модели показано на рис. 2.1.

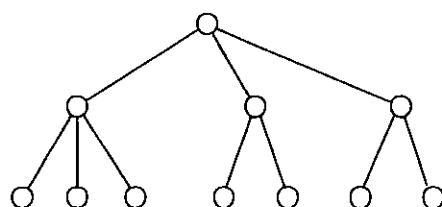


Рис. 2.1. Представление связей в иерархической модели

Для описания структуры (схемы) иерархической БД на некотором языке программирования используется тип данных «дерево».

Тип «дерево» схож с типами данных «структура» языков программирования ПЛ/1 и С и «запись» языка Паскаль. В них допускается вложенность типов, каждый из которых находится на некотором уровне.

Тип «дерево» является составным. Он включает в себя подтипы («поддеревья»), каждый из которых, в свою очередь, является типом «дерево». Каждый из типов «дерево» состоит из одного «корневого» типа и упорядоченного набора (возможно, пустого) подчиненных типов. Каждый из элементарных типов, включенных в тип «дерево», является простым или составным типом «запись». Простая «запись» состоит из одного типа, например числового, а составная «запись» объединяет некоторую совокупность типов, например, целое, строку символов и указатель (ссылку). Пример типа «дерево» как совокупности типов показан на рис. 2.2.



Рис. 2.2. Пример типа «дерево»

Корневым называется тип, который имеет подчиненные типы и сам не является подтипом. **Подчиненный тип** (подтип) является *потомком* по отношению к типу, который выступает для него в роли предка (родителя). Потомки одного и того же типа являются *близнецами* по отношению друг к другу.

В целом тип «дерево» представляет собой иерархически организованный набор типов «запись».

Иерархическая БД представляет собой упорядоченную совокупность экземпляров данных типа «дерево» (деревьев), содержащих экземпляры типа «запись» (записи). Часто отношения родства между типами переносят на отношения между самими записями. Поля записей хранят собственно числовые или символьные значения, составляющие основное содержание БД. Обход всех элементов иерархической БД обычно производится сверху вниз и слева направо.

В иерархических СУБД может использоваться терминология, отличающаяся от приведенной. Так, в системе IMS понятию «запись» соответствует

термин «сегмент», а под «записью БД» понимается вся совокупность записей, относящаяся к одному экземпляру типа «дерево».

Данные в базе с приведенной схемой (рис. 2.2) могут выглядеть, например, как показано на рис. 2.3.

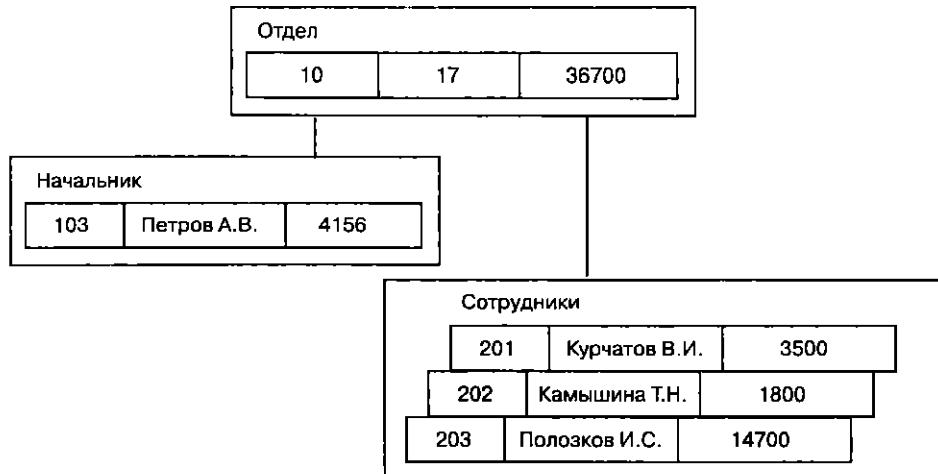


Рис. 2.3. Данные в иерархической базе

Для организации *физического* размещения иерархических данных в памяти ЭВМ могут использоваться следующие группы методов:

- представление линейным списком с последовательным распределением памяти (адресная арифметика, левосписковые структуры);
- представление связными линейными списками (методы, использующие указатели и справочники).

К основным операциям манипулирования иерархически организованными данными относятся следующие:

- поиск указанного экземпляра БД (например, дерева со значением 10 в поле Отд_номер);
- переход от одного дерева к другому;
- переход от одной записи к другой внутри дерева (например, к следующей записи типа Сотрудники);
- вставка новой записи в указанную позицию;
- удаление текущей записи и т. д.

В соответствии с определением типа «дерево», можно заключить, что между предками и потомками автоматически поддерживается контроль целостности связей. Основное правило контроля целостности формулируется следующим образом: потомок не может существовать без родителя, а у некоторых родителей может не быть потомков. Механизмы поддержания целостности

связей между записями различных деревьев отсутствуют.

К достоинствам иерархической модели данных относятся эффективное использование памяти ЭВМ и неплохие показатели времени выполнения основных операций над данными. Иерархическая модель данных удобна для работы с иерархически упорядоченной информацией.

Недостатком иерархической модели является ее громоздкость для обработки информации с достаточно сложными логическими связями, а также сложность понимания для обычного пользователя.

На иерархической модели данных основано сравнительно ограниченное количество СУБД, в числе которых можно назвать зарубежные системы IMS, PC/Focus, Team-Up и Data Edge, а также отечественные системы Ока, ИНЭС и МИРИС.

2.2. Сетевая модель

Сетевая модель данных позволяет отображать разнообразные взаимосвязи элементов данных в виде произвольного графа, обобщая тем самым иерархическую модель данных (рис. 2.4). Наиболее полно концепция сетевых БД впервые была изложена в Предложениях группы КОДАСИЛ (KODASYL).

Для описания схемы сетевой БД используется две группы типов: «запись» и «связь». Тип «связь» определяется для двух типов «запись»: предка и потомка. Переменные типа «связь» являются экземплярами связей.

Сетевая БД состоит из набора записей и набора соответствующих связей. На формирование связи особых ограничений не накладывается. Если в иерархических структурах запись-потомок могла иметь только одну запись-предка, то в сетевой модели данных запись-потомок может иметь произвольное число записей-предков (сводных родителей).

Рис. 2.4. Представление связей в сетевой модели

Пример схемы простейшей сетевой БД показан на рис. 2.5. Типы связей здесь обозначены надписями на соединяющих типы записей линиях.

В различных СУБД сетевого типа для обозначения одинаковых по сути понятий зачастую используются различные термины. Например, такие как элементы и агрегаты данных, записи, наборы, области и т. д.

Физическое размещение данных в базах сетевого типа может быть организовано практически теми же методами, что и в иерархических базах данных.

К числу важнейших операций манипулирования данными баз сетевого типа можно отнести следующие:

- поиск записи в БД;

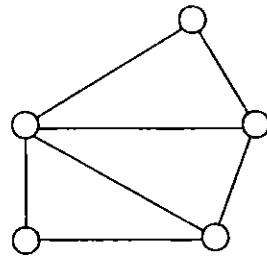




Рис. 2.5. Пример схемы сетевой БД

- переход от предка к первому потомку;
- переход от потомка к предку;
- создание новой записи;
- удаление текущей записи;
- обновление текущей записи;
- включение записи в связь;
- исключение записи из связи;
- изменение связей и т. д.

Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности. В сравнении с иерархической моделью сетевая модель предоставляет большие возможности в смысле допустимости образования произвольных связей.

Недостатком сетевой модели данных является высокая сложность и жесткость схемы БД, построенной на ее основе, а также сложность для понимания и выполнения обработки информации в БД обычным пользователем. Кроме того, в сетевой модели данных ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями.

Системы на основе сетевой модели не получили широкого распространения на практике. Наиболее известными сетевыми СУБД являются следующие: IDMS, db_VistaIII, СЕТЬ, СЕТОР и КОМПАС.

2.3. Реляционная модель

Реляционная модель данных предложена сотрудником фирмы IBM Эдгэром Коддом и основывается на понятии отношение (relation).

Отношение представляет собой множество элементов, называемых кортежами. Подробно теоретическая основа реляционной модели данных рассматривается в следующем разделе. Наглядной формой представления отношения является привычная для человеческого восприятия двумерная таблица.

Таблица имеет строки (записи) и столбцы (колонки). Каждая строка таблицы имеет одинаковую структуру и состоит из полей. Строкам таблицы соответствуют кортежи, а столбцам – атрибуты отношения.

С помощью одной таблицы удобно описывать простейший вид связей между данными, а именно деление одного объекта (явления, сущности, системы и проч.), информации о котором хранится в таблице, на множество подобъектов, каждому из которых соответствует строка или запись таблицы. При этом каждый из подобъектов имеет одинаковую структуру или свойства, описываемые соответствующими значениями полей записей. Например, таблица может содержать сведения о группе обучаемых, о каждом из которых известны следующие характеристики: фамилия, имя и отчество, пол, возраст и образование. Поскольку в рамках одной таблицы не удается описать более сложные логические структуры данных из предметной области, применяют *связывание* таблиц.

Физическое размещение данных в реляционных базах на внешних носителях легко осуществляется с помощью обычных файлов.

Достоинство реляционной модели данных заключается в простоте, понятности и удобстве физической реализации на ЭВМ. Именно простота и понятность для пользователя явились основной причиной их широкого использования. Проблемы же эффективности обработки данных этого типа оказались технически вполне разрешимыми.

Основными недостатками реляционной модели являются следующие: отсутствие стандартных средств идентификации отдельных записей и сложность описания иерархических и сетевых связей.

Примерами зарубежных реляционных СУБД для ПЭВМ являются следующие: dBaseII Plus и dBase IY (фирма Ashton-Tate), DB2 (IBM), R:BASE (Microrim), FoxPro ранних версий и FoxBase (Fox Software), Paradox и dBASE for Windows (Borland), FoxPro более поздних версий, Visual FoxPro и Access (Microsoft), Clarion (Clarion Software), Ingres (ASK Computer Systems) и Oracle (Oracle).

К отечественным СУБД реляционного типа относятся системы: ПАЛЬМА (ИКАН УССР), а также система NuTech (МИФИ).

Заметим, что последние версии реляционных СУБД имеют некоторые свойства объектно-ориентированных систем. Такие СУБД часто называют объектно-реляционными. Примером такой системы можно считать продукты Oracle 8.x. Системы предыдущих версий вплоть до Oracle 7.x считаются «чисто» реляционными.

2.4. Постреляционная модель

Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц. Это означает, что информация в таб-

лице представляется в первой нормальной форме (подраздел 5.2). Существует ряд случаев, когда это ограничение мешает эффективной реализации приложений.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Постреляционная модель данных допускает многозначные поля — поля, значения которых состоят из подзначений. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу.

На рис. 2.6 на примере информации о накладных и товарах для сравнения приведено представление одних и тех же данных с помощью реляционной (а) и постреляционной (б) моделей. Таблица INVOICES (накладные) содержит данные о номерах накладных (INVNO) и номерах покупателей (CUSTNO). В таблице INVOICE.ITEMS (накладные-товары) содержатся данные о каждой из накладных: номер накладной (INVNO), название товара (GOODS) и количество товара (QTY). Таблица INVOICES связана с таблицей INVOICE.ITEMS по полю INVNO.

Как видно из рисунка, по сравнению с реляционной моделью в постреляционной модели данные хранятся более эффективно, а при обработке не требуется выполнять операцию соединения данных из двух таблиц. Для доказательства на рис. 2.7 приводятся примеры операторов SELECT выбора данных из всех полей базы на языке SQL для реляционной (а) и постреляционной (б) моделей.

Помимо обеспечения вложенности полей постреляционная модель поддерживает ассоциированные многозначные поля (множественные группы). Совокупность ассоциированных полей называется **ассоциацией**. При этом в строке первое значение одного столбца ассоциации соответствует первым значениям всех других столбцов ассоциации. Аналогичным образом связаны все вторые значения столбцов и т. д.

На длину полей и количество полей в записях таблицы не накладывается требование постоянства. Это означает, что структура данных и таблиц имеет большую гибкость.

Поскольку постреляционная модель допускает хранение в таблицах не-нормализованных данных, возникает проблема обеспечения целостности и непротиворечивости данных. Эта проблема решается включением в СУБД механизмов, подобных хранимым процедурам в клиент-серверных системах.

Для описания функций контроля значений в полях имеется возможность создавать процедуры (коды конверсии и коды корреляции), автоматически вызываемые до или после обращения к данным. Коды корреляции выполняются сразу после чтения данных, перед их обработкой. Коды конверсии, наоборот, выполняются после обработки данных.

а)

INVOICES

INVNO	CUSTNO
0373	8723
8374	8232
7364	8723

INVOICE.ITEMS

INVNO	GOODS	QTY
0373	Сыр	3
0373	Рыба	2
8374	Лимонад	1
8374	Сок	6
8374	Печенье	2
7364	Йогурт	1

б)

INVOICES

INVNO	CUSTNO	GOODS	QTY
0373	8723	Сыр	3
		Рыба	2
8374	8232	Лимонад	1
		Сок	6
		Печенье	2
		Йогурт	1

Рис. 2.6. Структуры данных реляционной и постреляционной моделей

Достоинством постреляционной модели является возможность представления совокупности связанных реляционных таблиц одной постреляционной таблицей. Это обеспечивает высокую наглядность представления информации и повышение эффективности ее обработки.

Недостатком постреляционной модели является сложность решения проблем обеспечения целостности и непротиворечивости хранимых данных.

```
a)
SELECT
    INVOICES.INVNO, CUSTNO, GOODS, QTY
FROM
    INVOICES, INVOICE.ITEMS
WHERE
    INVOICES.INVNO=INVOICE.ITEMS.INVNO;

b)
SELECT
    INVNO, CUSTNO, GOODS, QTY
FROM
    INVOICES;
```

Рис. 2.7. Операторы SQL для реляционной и постреляционной моделей

Рассмотренная нами постреляционная модель данных поддерживается СУБД uniVers. К числу других СУБД, основанных на постреляционной модели данных, относятся также системы Bubba и Dasdb.

2.5. Многомерная модель

Многомерный подход к представлению данных в базе появился практически одновременно с реляционным, но реально работающих многомерных СУБД (МСУБД) до настоящего времени было очень мало. С середины 90-х годов интерес к ним стал приобретать массовый характер.

Толчком послужила в 1993 году программная статья одного из основоположников реляционного подхода Э. Кодда. В ней сформулированы 12 основных требований к системам класса OLAP (OnLine Analytical Processing – оперативная аналитическая обработка), важнейшие из которых связаны с возможностями концептуального представления и обработки многомерных данных. Многомерные системы позволяют оперативно обрабатывать информацию для проведения анализа и принятия решения.

В развитии концепций ИС можно выделить следующие два направления:

- системы оперативной (транзакционной) обработки;
- системы аналитической обработки (системы поддержки принятия решений).

Реляционные СУБД предназначались для информационных систем *оперативной* обработки информации и в этой области были весьма эффективны. В системах аналитической обработки они показали себя несколько неповоротливыми и недостаточно гибкими. Более эффективными здесь оказываются многомерные СУБД (МСУБД).

Многомерные СУБД являются узкоспециализированными СУБД, предназначенными для интерактивной аналитической обработки информации. Раскроем основные понятия, используемые в этих СУБД: агрегируемость, историчность и прогнозируемость данных.

Агрегируемость данных означает рассмотрение информации на различных уровнях ее обобщения. В информационных системах степень детальности представления информации для пользователя зависит от его уровня: аналитик, пользователь-оператор, управляющий, руководитель.

Историчность данных предполагает обеспечение высокого уровня статичности (неизменности) собственно данных и их взаимосвязей, а также обязательность привязки данных ко времени.

Статичность данных позволяет использовать при их обработке специализированные методы загрузки, хранения, индексации и выборки.

Временная привязка данных необходима для частого выполнения запросов, имеющих значения времени и даты в составе выборки. Необходимость упорядочения данных по времени в процессе обработки и представления данных пользователю накладывает требования на механизмы хранения и доступа к информации. Так, для уменьшения времени обработки запросов желательно, чтобы данные всегда были отсортированы в том порядке, в котором они наиболее часто запрашиваются.

Прогнозируемость данных подразумевает задание функций прогнозирования и применение их к различным времененным интервалам.

Многомерность модели данных означает не многомерность визуализации цифровых данных, а многомерное логическое представление структуры информации при описании и в операциях манипулирования данными.

По сравнению с реляционной моделью многомерная организация данных обладает более высокой *наглядностью и информативностью*. Для иллюстрации на рис. 2.8 приведены реляционное (а) и многомерное (б) представления одних и тех же данных об объемах продаж автомобилей.

Если речь идет о многомерной модели с мерностью больше двух, то не обязательно визуально информация представляется в виде многомерных объектов (трех-, четырех- и более мерных гиперкубов). Пользователю и в этих случаях более удобно иметь дело с двухмерными таблицами или графиками. Данные при этом представляют собой «вырезки» (точнее, «срезы») из многомерного хранилища данных, выполненные с разной степенью детализации.

Рассмотрим основные понятия многомерных моделей данных, к числу которых относятся измерение и ячейка.

Измерение (Dimension) – это множество однотипных данных, образующих одну из граней гиперкуба. Примерами наиболее часто используемых временных измерений являются Дни, Месяцы, Кварталы и Годы. В качестве географических измерений широко употребляются Города, Районы, Ре-

а)

Модель	Месяц	Объем
«Жигули»	июнь	12
«Жигули»	июль	24
«Жигули»	август	5
«Москвич»	июнь	2
«Москвич»	июль	18
«Волга»	июль	19

б)

Модель	Июнь	Июль	Август
«Жигули»	12	24	5
«Москвич»	2	18	No
«Волга»	No	19	No

Рис. 2.8. Реляционное и многомерное представление данных

гионы и Страны. В многомерной модели данных измерения играют роль индексов, служащих для идентификации конкретных значений в ячейках гиперкуба.

Ячейка (Cell) или показатель — это поле, значение которого однозначно определяется фиксированным набором измерений. Тип поля чаще всего определен как цифровой. В зависимости от того, как формируются значения некоторой ячейки, обычно она может быть переменной (значения изменяются и могут быть загружены из внешнего источника данных или сформированы программно) либо формулой (значения, подобно формульным ячейкам электронных таблиц, вычисляются по заранее заданным формулам).

В примере на рис. 2.8, б каждое значение ячейки Объем продаж однозначно определяется комбинацией временного измерения (Месяц продаж) и модели автомобиля. На практике зачастую требуется большее количество измерений. Пример трехмерной модели данных приведен на рис. 2.9.

В существующих МСУБД используются два основных варианта (схемы) организации данных: гиперкубическая и поликубическая.

В поликубической схеме предполагается, что в БД может быть определено несколько гиперкубов с различной размерностью и с различными измерени-

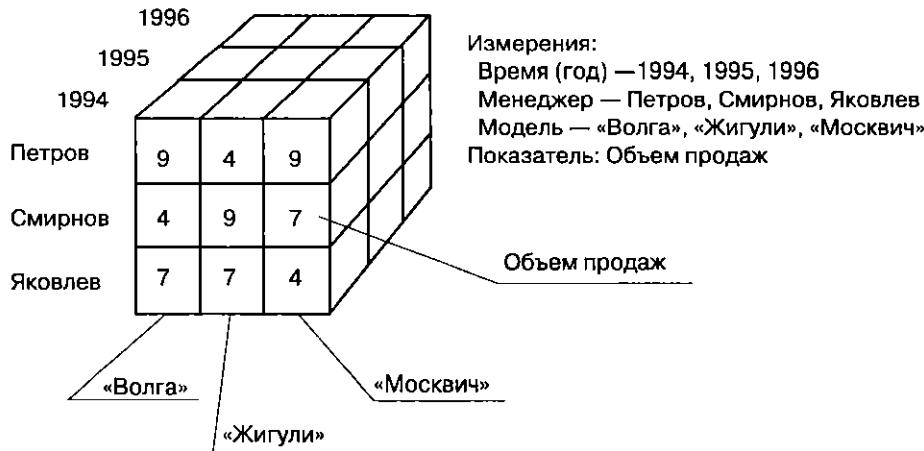


Рис. 2.9. Пример трехмерной модели

ями в качестве граней. Примером системы, поддерживающей поликубический вариант БД, является сервер Oracle Express Server.

В случае *гиперкубической* схемы предполагается, что все показатели определяются одним и тем же набором измерений. Это означает, что при наличии нескольких гиперкубов БД все они имеют одинаковую размерность и совпадающие измерения. Очевидно, в некоторых случаях информация в БД может быть избыточной (если требовать обязательное заполнение ячеек).

В случае многомерной модели данных применяется ряд специальных операций, к которым относятся: формирование «среза», «вращение», агрегация и детализация.

«Срез» (Slice) представляет собой подмножество гиперкуба, полученное в результате фиксации одного или нескольких измерений. Формирование «срезов» выполняется для ограничения используемых пользователем значений, так как все значения гиперкуба практически никогда одновременно не используются. Например, если ограничить значения измерения Модель автомобиля в гиперкубе (рис. 2.9) маркой «Жигули», то получится двухмерная таблица продаж этой марки автомобиля различными менеджерами по годам.

Операция «вращение» (Rotate) применяется при двухмерном представлении данных. Суть ее заключается в изменении порядка измерений при визуальном представлении данных. Так, «вращение» двухмерной таблицы, показанной на рис. 2.8 б, приведет к изменению ее вида таким образом, что по оси X будет марка автомобиля, а по оси Y — время.

Операцию «вращение» можно обобщить и на многомерный случай, если под ней понимать процедуру изменения порядка следования измерений.

В простейшем случае, например, это может быть взаимная перестановка двух произвольных измерений.

Операции «агрегация» (Drill Up) и «детализация» (Drill Down) означают соответственно переход к более общему и к более детальному представлению информации пользователю из гиперкуба.

Для иллюстрации смысла операции «агрегация» предположим, что у нас имеется гиперкуб, в котором помимо измерений гиперкуба, приведенного на рис. 2.9, имеются еще измерения: Подразделение, Регион, Фирма, Страна. Заметим, что в этом случае в гиперкубе существует иерархия (снизу вверх) отношений между измерениями: Менеджер, Подразделение, Регион, Фирма, Страна.

Пусть в описанном гиперкубе определено, насколько успешно в 1995 году менеджер Петров продавал автомобили «Жигули» и «Волга». Тогда, поднимаясь на уровень выше по иерархии, с помощью операции «агрегация» можно выяснить, как выглядит соотношение продаж этих же моделей на уровне подразделения, где работает Петров.

Основным достоинством многомерной модели данных является удобство и эффективность аналитической обработки больших объемов данных, связанных со временем. При организации обработки аналогичных данных на основе реляционной модели происходит нелинейный рост трудоемкости операций в зависимости от размерности БД и существенное увеличение затрат оперативной памяти на индексацию.

Недостатком многомерной модели данных является ее громоздкость для простейших задач обычной оперативной обработки информации.

Примерами систем, поддерживающих многомерные модели данных, являются Essbase (Arbor Software), Media Multi-matrix (Speedware), Oracle Express Server (Oracle) и Cache (InterSystems). Некоторые программные продукты, например Media/MR (Speedware), позволяют одновременно работать с многомерными и с реляционными БД. В СУБД Cache, в которой внутренней моделью данных является многомерная модель, реализованы три способа доступа к данным: прямой (на уровне узлов многомерных массивов), объектный и реляционный.

2.6. Объектно-ориентированная модель

В объектно-ориентированной модели при представлении данных имеется возможность идентифицировать отдельные записи базы. Между записями базы данных и функциями их обработки устанавливаются взаимосвязи с помощью механизмов, подобных соответствующим средствам в объектно-ориентированных языках программирования.

Стандартизованная объектно-ориентированной модель описана в рекомендациях стандарта ODMG-93 (Object Database Management Group — группа управления объектно-ориентированными базами данных). Реализовать в полном объе-

ме рекомендации ODMG-93 пока не удается. Для иллюстрации ключевых идей рассмотрим несколько упрощенную модель объектно-ориентированной БД.

Структура объектно-ориентированной БД графически представима в виде дерева, узлами которого являются объекты. Свойства объектов описываются некоторым стандартным типом (например, строковым — string) или типом, конструируемым пользователем (определяется как class).

Значением свойства типа string является строка символов. Значение свойства типа class есть объект, являющийся экземпляром соответствующего класса. Каждый объект-экземпляр класса считается потомком объекта, в котором он определен как свойство. Объект-экземпляр класса принадлежит своему классу и имеет одного родителя. Родовые отношения в БД образуют связную иерархию объектов.

Пример логической структуры объектно-ориентированной БД библиотечного дела приведен на рис. 2.10.

Здесь объект типа БИБЛИОТЕКА является родительским для объектов-экземпляров классов АБОНЕНТ, КАТАЛОГ и ВЫДАЧА. Различные объекты типа КНИГА могут иметь одного или разных родителей. Объекты типа КНИГА, имеющие одного и того же родителя, должны различаться по крайней мере инвентарным номером (уникален для каждого экземпляра книги), но имеют одинаковые значения свойств *isbn*, *удк*, *название* и *автор*.

Логическая структура объектно-ориентированной БД внешне похожа на структуру иерархической БД. Основное отличие между ними состоит в методах манипулирования данными.

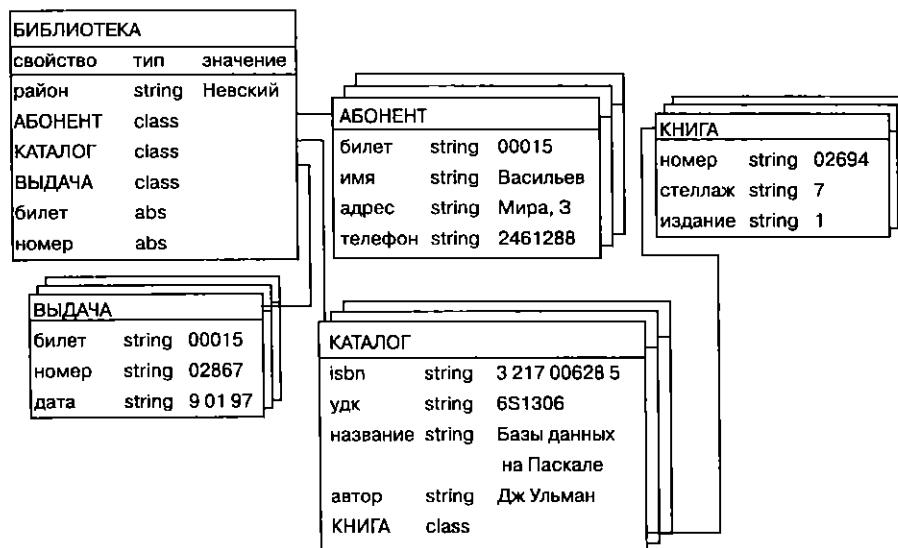


Рис. 2.10. Логическая структура БД библиотечного дела

Для выполнения действий над данными в рассматриваемой модели БД применяются логические операции, усиленные объектно-ориентированными механизмами инкапсуляции, наследования и полиморфизма. Ограниченно могут применяться операции, подобные командам SQL (например, для создания БД).

Создание и модификация БД сопровождается автоматическим формированием и последующей корректировкой индексов (индексных таблиц), содержащих информацию для быстрого поиска данных.

Рассмотрим кратко понятия инкапсуляции, наследования и полиморфизма применительно к объектно-ориентированной модели БД.

Инкапсуляция ограничивает область видимости имени свойства пределами того объекта, в котором оно определено. Так, если в объект типа КАТАЛОГ добавить свойство, задающее телефон автора книги и имеющее название *телефон*, то мы получим одноименные свойства у объектов АБОНЕНТ и КАТАЛОГ. Смысл такого свойства будет определяться тем объектом, в который оно инкапсулировано.

Наследование, наоборот, распространяет область видимости свойства на всех потомков объекта. Так, всем объектам типа КНИГА, являющимся потомками объекта типа КАТАЛОГ, можно присвоить свойства объекта-родителя: *isbn*, *удк*, *название* и *автор*. Если необходимо расширить действие механизма наследования на объекты, не являющиеся непосредственными родственниками (например, между двумя потомками одного родителя), то в их общем предке определяется абстрактное свойство типа *abs*. Так, определение абстрактных свойств *билет* и *номер* в объекте БИБЛИОТЕКА приводит к наследованию этих свойств всеми дочерними объектами АБОНЕНТ, КНИГА и ВЫДАЧА. Не случайно поэтому значения свойства *билет* классов АБОНЕНТ и ВЫДАЧА, показанных на рисунке, будут одинаковыми – 00015.

Полиморфизм в объектно-ориентированных языках программирования означает способность одного и того же программного кода работать с разнотипными данными. Другими словами, он означает допустимость в объектах разных типов иметь методы (процедуры или функции) с одинаковыми именами. Во время выполнения объектной программы одни и те же методы оперируют с разными объектами в зависимости от типа аргумента. Применительно к нашей объектно-ориентированной БД полиморфизм означает, что объекты класса КНИГА, имеющие разных родителей из класса КАТАЛОГ, могут иметь разный набор свойств. Следовательно, программы работы с объектами класса КНИГА могут содержать полиморфный код.

Поиск в объектно-ориентированной БД состоит в выяснении сходства между объектом, задаваемым пользователем, и объектами, хранящимися в БД. Определяемый пользователем объект, называемый объектом-целью (свойство объекта имеет тип *goal*), в общем случае может представлять собой подмножество всей хранимой в БД иерархии объектов. Объект-цель, а также резуль-

тат выполнения запроса могут храниться в самой базе. Пример запроса о номерах читательских билетов и именах абонентов, получавших в библиотеке хотя бы одну книгу, показан на рис. 2.11.



Рис. 2.11. Фрагмент БД с объектом-целью

Основным достоинством объектно-ориентированной модели данных в сравнении с реляционной является возможность отображения информации о сложных взаимосвязях объектов. Объектно-ориентированная модель данных позволяет идентифицировать отдельную запись базы данных и определять функции их обработки.

Недостатками объектно-ориентированной модели являются высокая понятийная сложность, неудобство обработки данных и низкая скорость выполнения запросов.

В 90-е годы существовали экспериментальные прототипы объектно-ориентированных систем управления базами данных. В настоящее время такие системы получили достаточно широкое распространение, в частности, к ним относятся следующие СУБД: G-Base (Grapael), GemStone (Servio-Logic совместно с OGI), Statice (Symbolics), ObjectStore (Object Design), Objectivity /DB (Objectivity), Versant (Versant Technologies), O2 (Ardent Software), ODB-Jupiter (научно-производственный центр «Интелтек Плюс»), а также Iris, Orion и Postgres.

2.7. Типы данных

Основные типы данных СУБД

Первоначально СУБД применялись преимущественно для решения финансово-экономических задач. При этом, независимо от модели представления, в базах данных использовались следующие основные типы данных:

- числовые. В качестве подтипов числовых данных часто используются

целочисленные, денежные (финансовые) и обычные вещественные. Примеры значений данных: 0.43, 328, 2E+5;

- символьные (алфавитно-цифровые). Примеры значений данных: «пятница», «строка», «программист»;
- логические, принимающие значения «истина» (true) и «ложь» (false);
- даты, задаваемые с помощью специального типа «Дата» или как обычные символьные данные. Примеры значений данных: 1.12.97, 2/23/1999.

В разных СУБД эти типы могли несущественно отличаться друг от друга по названию, диапазону значений и виду представления. С расширением области применения персональных компьютеров стали появляться специализированные системы обработки данных, например, геоинформационные, обработки видеоизображений и т. д. В ответ на это разработчики СУБД стали вводить в них поддержку новых типов данных. К числу сравнительно новых типов данных можно отнести следующие:

- временные и дата-временные, предназначенные для хранения информации о времени и/или дате. Примеры значений данных: 31.01.85 (дата), 9:10:03 (время), 6.03.1960 12:00 (дата и время);
- символьные переменной длины, предназначенные для хранения текстовой информации большой длины, например, документа;
- двоичные, предназначенные для хранения графических объектов, аудио- и видеинформации, пространственной, хронологической и другой специальной информации. Двоичные данные часто называют *мультимедиа-данными*. Например, в MS Access таким типом является тип данных «Поле объекта OLE», который позволяет хранить в БД графические данные в формате BMP (Bitmap) и автоматически их отображать при работе с БД;
- гиперссылки (hyperlinks), предназначенные для хранения ссылок на различные ресурсы (узлы, файлы, документы и т. д.), находящиеся вне базы данных, например, в сети Internet, корпоративной сети intranet или на жестком диске компьютера. Примеры значений данных: <http://www.chat.ru>, <ftp://chance4u.teens.com>;
- данные в XML формате.

Технология OLE (Object Linking and Embedding) реализует такой механизм связывания и встраивания объектов, при котором для обработки объекта вызывается приложение, в котором этот объект создавался.

Мультимедиа-данные

Слово мультимедиа (multimedia) стало популярным в компьютерной области в 90-х годах. Точного перевода его с английского языка на русский не существует, сравнительно близки следующие варианты перевода: «многосредность» или «множество сред». Под средами здесь понимаются данные различной природы: звуковые, видео-, графические, текстовые, с различными эффектами отображения на экране (анимацией) и т. д.

В широком смысле термин мультимедиа означает совокупность технологий производства и применения различных аппаратных и программных средств для ПЭВМ, позволяющих поддерживать работу компьютера с перечисленными видами информации.

Мультимедиа-средства нужны, чтобы существенно оживить процедуру общения пользователя с компьютером. Использование технологии мультимедиа позволяет расширить область применения ПЭВМ и поднять на более высокий качественный уровень решение традиционных задач.

Говоря о мультимедиа-возможностях реляционных систем, как получивших наибольшее распространение, отметим, что основная их роль – хранить мультимедиа-данные. Порождать и видоизменять сами данные в настоящее время могут только мультимедиа-программы, каковыми СУБД не являются.

В реляционных системах основным местом хранения данных, в том числе и мультимедиа-данных, является таблица. Для обеспечения хранения мультимедиа-данных в структуре таблицы должны быть предусмотрены соответствующие поля. Кроме того, мультимедиа-данные могут храниться в экранных формах и отчетах.

Главное отличие названных способов состоит в том, что в первом случае мультимедиа-данные связываются с каждой записью базы, а во втором случае мультимедиа-данные включаются в экранную форму или отчет один раз. Так, при просмотре БД с помощью экранной формы связанные с записями мультимедиа-данные при изменении текущей записи изменяются, а мультимедиа-данные самой экранной формы – остаются без изменения. Мультимедиа-данные включают в экранные формы и отчеты для повышения их наглядности при отображении на экране (рис. 2.12).

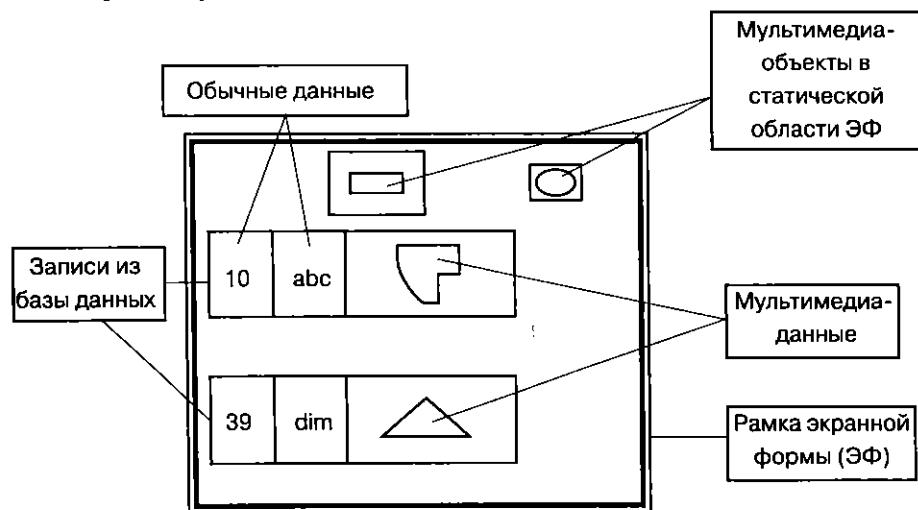


Рис. 2.12. Мультимедиа-данные в экранной форме

В различных СУБД применяются разные механизмы поддержания мультимедиа-данных. Чаще всего для их размещения хранения используются поля хранения двоичных объектов или так называемые BLOB-поля (Binary Large OBject – большие двоичные объекты). Поскольку мультимедиа-данные могут иметь различные виды (аудио-, видео-, графическая и т. д. информация), а в рамках каждого вида – разные форматы (например, для хранения графической информации используются файлы с расширениями: bmp, psx, tif, gif, eps и т. д.), удобным способом привязки их к средствам обработки оказывается упоминавшийся ранее механизм OLE. В связи с этим одним из наиболее распространенных типов BLOB-полей являются OLE-поля.

К примеру, в MS Access поддерживается OLE-поле, в системе Paradox можно создавать также поля типа graphic и binary (в поле типа graphic информация только интерпретируется и отсутствует связь с приложением; информация из поля типа binary не отображается на экране, так как СУБД ее не интерпретирует).

Поскольку для работы с мультимедиа-информацией в общем случае могут использоваться как специализированные средства ее обработки и хранения, так и реляционные системы, то перед пользователем может встать вопрос: «А что же лучше?». Существенную роль в ответе на этот вопрос, по всей видимости, будет играть специфика решаемой задачи.

В качестве общей рекомендации при выборе того или иного программного продукта можно принять следующее. Если в прикладной задаче используется разного рода информация и доля обычной символьно-числовой информации велика, то лучше остановить свой выбор на реляционной системе (при наличии в ней достаточно развитых средств поддержки мультимедиа-данных). Причина такого решения очевидна – реляционные системы сейчас являются достаточно проработанными и эффективными в применении. В случае преобладания потребностей в обработке двоичной информации с использованием специальных методик, алгоритмов и интерфейсов, скорее всего следует остановить свой выбор на специализированных средствах работы с мультимедиа-данными.

Контрольные вопросы и задания

1. Перечислите классические и современные модели представления данных.
2. Укажите достоинства и недостатки иерархической модели данных.
3. Как организуется физическое размещение данных в БД иерархического типа?
4. Охарактеризуйте сетевую модель данных.
5. Охарактеризуйте реляционную модель данных.
6. В чем отличие между постреляционной и реляционной моделями данных?
7. Укажите достоинства и недостатки постреляционной модели.

8. Охарактеризуйте многомерную модель данных.
9. Назовите и поясните смысл операций, выполнимых над данными в случае многомерной модели.
10. Дайте определение и приведите примеры проявления принципов инкапсуляции, полиморфизма и наследования применительно к объектно-ориентированным базам данных.
11. Укажите достоинства и недостатки объектно-ориентированной модели представления данных.
12. Охарактеризуйте типы данных, используемые в современных СУБД.
13. Можно ли хранить и просматривать рисунки в БД СУБД MS Access?
14. Охарактеризуйте мультимедиа-возможности реляционных систем.

Литература

1. *Биттинер Ю.* Интербаза — система управления базами данных нового поколения // Вычисл. Техника соцстран. Вып. 26. С. 136–143.
2. *Бородаев В. А., Кустов В. Н.* Банки и базы данных. Уч. пособие. Л.: ВИКИ, 1989.
3. *Васильев В.* Объектно-ориентированная БД: взгляд изнутри // Компьютеры + Программы, № 3 (36), 1997. С. 45–49.
4. *Ким Е., Шабаев И., Бычков В.* Проектирование трехмерных баз данных в СУБД uniVerse // Системы Управления Базами Данных, № 3, 1996. С. 66-76.
5. Основы современных компьютерных технологий: Учебник / Под ред. проф. Хомоненко А. Д. Авторы: Брякалов Г. А., Войцеховский С. В., Воробьев Е. Г., Гофман В. Э., Гридин В. В., Дрюков Ю. П., Замула А. А., Захаров А. И., Компаниец Р. И., Липецких А. Г., Рыжиков Ю. И., Хомоненко А. Д., Цыганков В. М. – СПб: КОРОНА прнт, 2005. – 672 с.
6. *Сахаров А. А.* Принципы проектирования и использования многомерных баз данных (на примере Oracle Express Server) // Системы Управления Базами Данных, № 3, 1996. С. 44–59.
7. Системы управления базами данных и знаний: Справ. изд. / Наумов А. Н., Венгров А. М., Иванов В. К. и др.; Под ред. А. Н. Наумова. М.: Финансы и статистика, 1991.
8. *Четвериков В. Н.* и др. Базы и банки данных: Учебник для вузов по спец. «АСУ» / Под ред. В. Н. Четверикова. М.: Высп. шк., 1987.
9. *Фролов А. В., Фролов Г. В.* Мультимедиа для Windows. Руководство для программиста. – М.: ДИАЛОГ-МИФИ, 1995. – 284 с.

3. Реляционная модель данных

В разделе рассматривается наиболее распространенная реляционная модель представления данных; дается определение реляционной модели и характеристика ее элементов; описываются индексирование, связывание таблиц и контроль целостности связей; рассматриваются теоретические основы построения языков запросов: реляционная алгебра и реляционное исчисление; а также дается характеристика языков QBE и SQL, формат операторов и примеры построения запросов с их помощью.

3.1. Определение реляционной модели

Реляционная модель данных (РМД) некоторой предметной области представляет собой набор отношений, изменяющихся во времени. При создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними. Элементы РМД и формы их представления приведены в табл. 3.1.

Таблица 3.1
Элементы реляционной модели

Элемент реляционной модели	Форма представления
Отношение	Таблица
Схема отношения	Строка заголовков столбцов таблицы (заголовок таблицы)
Кортеж	Строка таблицы
Сущность	Описание свойств объекта
Атрибут	Заголовок столбца таблицы
Домен	Множество допустимых значений атрибута
Значение атрибута	Значение поля в записи
Первичный ключ	Один или несколько атрибутов
Тип данных	Тип значений элементов таблицы

Отношение является важнейшим понятием и представляет собой двумерную таблицу, содержащую некоторые данные.

Сущность есть объект любой природы, данные о котором хранятся в базе данных. Данные о сущности хранятся в отношении.

Атрибуты представляют собой свойства, характеризующие сущность. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок некоторого столбца таблицы.

Математически отношение можно описать следующим образом. Пусть даны n множеств $D_1, D_2, D_3, \dots, D_n$, тогда отношение R есть множество упорядоченных **кортежей** $\langle d_1, d_2, d_3, \dots, d_n \rangle$, где $d_k \in D_k$, d_k – **атрибут**, а D_k – **домен** отношения R .

На рис. 3.1 приведен пример представления отношения СОТРУДНИК.

В общем случае порядок кортежей в отношении, как и в любом множестве, не определен. Однако в реляционных СУБД для удобства кортежи все же упорядочиваются. Чаще всего для этого выбирают некоторый атрибут, по которому система автоматически сортирует кортежи по возрастанию или убыванию. Если пользователь не назначает атрибута упорядочения, система автоматически присваивает номер кортежам в порядке их ввода.

ФИО	Отдел	Должность	Д_рождения
Иванов И.И.	002	Начальник	27.09.51
Петров П.П.	001	Заместитель	15.04.55
Сидоров И.П.	002	Инженер	13.01.70

Рис. 3.1. Представление отношения СОТРУДНИК

Формально, если переставить атрибуты в отношении, то получается новое отношение. Однако в реляционных БД перестановка атрибутов не приводит к образованию нового отношения.

Домен представляет собой множество всех возможных значений определенного атрибута отношения. Отношение СОТРУДНИК включает 4 домена. Домен 1 содержит фамилии всех сотрудников, домен 2 – номера всех отделов

фирмы, *домен 3* – названия всех должностей, *домен 4* – даты рождения всех сотрудников. Каждый домен образует значения одного типа данных, например, числовые или символьные.

Отношение СОТРУДНИК содержит 3 кортежа. Кортеж рассматриваемого отношения состоит из 4 элементов, каждый из которых выбирается из соответствующего домена. Каждому кортежу соответствует строка таблицы (рис. 3.1).

Схема отношения (заголовок отношения) представляет собой список имен атрибутов. Например, для приведенного примера схема отношения имеет вид СОТРУДНИК(ФИО, Отдел, Должность, Д_Рождения). Множество собственно кортежей отношения часто называют **содержимым (телом) отношения**.

Первичным ключом (ключом отношения, ключевым атрибутом) называется атрибут отношения, однозначно идентифицирующий каждый из его кортежей. Например, в отношении СОТРУДНИК(ФИО, Отдел, Должность, Д_Рождения) ключевым является атрибут «ФИО». Ключ может быть **составным (сложным)**, то есть состоять из нескольких атрибутов.

Каждое отношение обязательно имеет комбинацию атрибутов, которая может служить ключом. Ее существование гарантируется тем, что отношение – это множество, которое не содержит одинаковых элементов – кортежей. То есть в отношении нет повторяющихся кортежей, а это значит, что по крайней мере вся совокупность атрибутов обладает свойством однозначной идентификации кортежей отношения. Во многих СУБД допускается создавать отношения, не определяя ключи.

Возможны случаи, когда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет все кортежи отношения. Все эти комбинации атрибутов являются **возможными ключами** отношения. Любой из возможных ключей может быть выбран как *первичный*.

Если выбранный первичный ключ состоит из минимально необходимого набора атрибутов, говорят, что он является **не избыточным**.

Ключи обычно используют для достижения следующих целей:

- 1) исключения дублирования значений в ключевых атрибутах (остальные атрибуты в расчет не принимаются);
- 2) упорядочения кортежей. Возможно упорядочение по возрастанию или убыванию значений всех ключевых атрибутов, а также смешанное упорядочение (по одним – возрастание, а по другим – убывание);
- 3) ускорения работы к кортежами отношения (подраздел 3.2);
- 4) организации связывания таблиц (подраздел 3.3).

Пусть в отношении R1 имеется *не ключевой* атрибут А, значения которого являются значениями *ключевого* атрибута В другого отношения R2. Тогда говорят, что атрибут А отношения R1 есть **внешний ключ**.

С помощью внешних ключей устанавливаются связи между отношениями. Например, имеются два отношения СТУДЕНТ(ФИО, Группа, Специальность) и ПРЕДМЕТ(Назв.Пр, Часы), которые связаны отношением СТУДЕНТ_ПРЕДМЕТ(ФИО, Назв.Пр, Оценка) (рис. 3.2). В связующем отношении атрибуты ФИО и Назв.Пр образуют составной ключ. Эти атрибуты представляют собой внешние ключи, являющиеся первичными ключами других отношений.

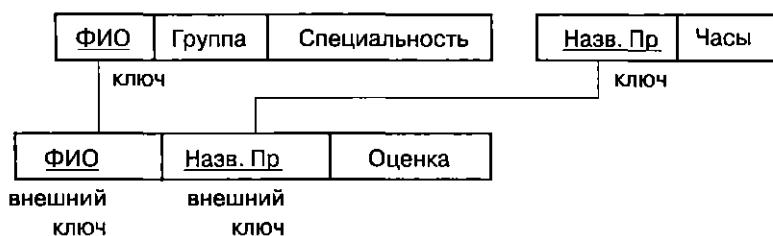


Рис. 3.2. Связь отношений

Реляционная модель накладывает на внешние ключи ограничение для обеспечения целостности данных, называемое *ссылочной целостностью*. Это означает, что каждому значению внешнего ключа должны соответствовать строки в связываемых отношениях.

Поскольку не всякой таблице можно поставить в соответствие отношение, приведем условия, выполнение которых позволяет таблицу считать отношением.

1. Все строки таблицы должны быть уникальны, то есть не может быть строк с одинаковыми первичными ключами.
2. Имена столбцов таблицы должны быть различны, а значения их простыми, то есть недопустима группа значений в одном столбце одной строки.
3. Все строки одной таблицы должны иметь одну структуру, соответствующую именам и типам столбцов.
4. Порядок размещения строк в таблице может быть произвольным.

Наиболее часто таблица с отношением размещается в отдельном файле. В некоторых СУБД одна отдельная таблица (отношение) считается базой данных. В других СУБД база данных может содержать несколько таблиц.

В общем случае можно считать, что БД включает одну или несколько таблиц, объединенных смысловым содержанием, а также процедурами контроля целостности и обработки информации в интересах решения некоторой прикладной задачи. Например, при использовании СУБД Microsoft Access в файле БД паряду с таблицами хранятся и другие объекты базы: запросы, отчеты, формы, макросы и модули.

Таблица данных обычно хранится на магнитном диске в отдельном файле операционной системы, поэтому по ее именованию могут существовать ограничения. Имена полей хранятся внутри таблиц. Правила их формирования определяются СУБД, которые, как правило, на длину полей и используемый алфавит серьезных ограничений не накладывают.

Если задаваемое таблицей отношение имеет ключ, то считается, что таблица тоже имеет ключ, и ее называют **ключевой** или **таблицей с ключевыми полями**.

У большинства СУБД файл таблицы включает управляющую часть (описание типов полей, имена полей и другая информация) и область размещения записей.

К отношениям можно применять систему операций, позволяющую получать одни отношения из других. Например, результатом запроса к реляционной БД может быть новое отношение, вычисленное на основе имеющихся отношений. Поэтому можно разделить обрабатываемые данные на хранимую и вычисляемую части.

Основной единицей обработки данных в реляционных БД является отношение, а не отдельные его кортежи (записи).

3.2. Индексирование

Как отмечалось выше, определение ключа для таблицы означает автоматическую сортировку записей, контроль отсутствия повторений значений в ключевых полях записей и повышение скорости выполнения операций поиска в таблице. Для реализации этих функций в СУБД применяют **индексирование**.

Термин «индекс» тесно связан с понятием «ключ», хотя между ними есть и некоторое отличие.

Под **индексом** понимают средство **ускорения** операции поиска записей в таблице, а следовательно, и других операций, использующих поиск: извлечение, модификация, сортировка и т. д. Таблицу, для которой используется индекс, называют **индексированной**.

Индекс выполняет роль **оглавления** таблицы, просмотр которого предшествует обращению к записям таблицы. В некоторых системах, например, Paradox, индексы хранятся в индексных файлах, хранимых отдельно от табличных файлов.

Варианты решения проблемы организации физического доступа к информации зависят в основном от следующих факторов:

- вида содержимого в поле ключа записей индексного файла;
- типа используемых ссылок (указателей) на запись основной таблицы;
- метода поиска нужных записей.

В поле *ключа индексного файла* можно хранить значения ключевых полей индексируемой таблицы либо свертку ключа (так называемый хеш-код). Преимущество хранения хеш-кода вместо значения состоит в том, что длина свертки независимо от длины исходного значения ключевого поля всегда имеет некоторую постоянную и достаточно малую величину (например, 4 байта), что существенно снижает время поисковых операций. Недостатком хеширования является необходимость выполнения операции свертки (требует определенного времени), а также борьба с возникновением коллизий (свертка различных значений может дать одинаковый хеш-код).

Для организации *ссылки* на запись таблицы могут использоваться три типа адресов: абсолютный (действительный), относительный и символьический (идентификатор).

На практике чаще всего используются *два метода поиска*: последовательный и бинарный (основан на делении интервала поиска пополам).

Проиллюстрируем организацию индексирования таблиц двумя схемами: одноуровневой и двухуровневой. При этом примем ряд предположений, обычно выполняемых в современных вычислительных системах. Пусть ОС поддерживает прямую организацию данных на магнитных дисках, основные таблицы и индексные файлы хранятся в отдельных файлах. Информация файлов хранится в виде совокупности блоков фиксированного размера, например целого числа кластеров.

При *одноуровневой схеме* в индексном файле хранятся короткие записи, имеющие два поля: поле содержимого старшего ключа (хеш-кода ключа) адресуемого блока и поле адреса начала этого блока (рис. 3.3).

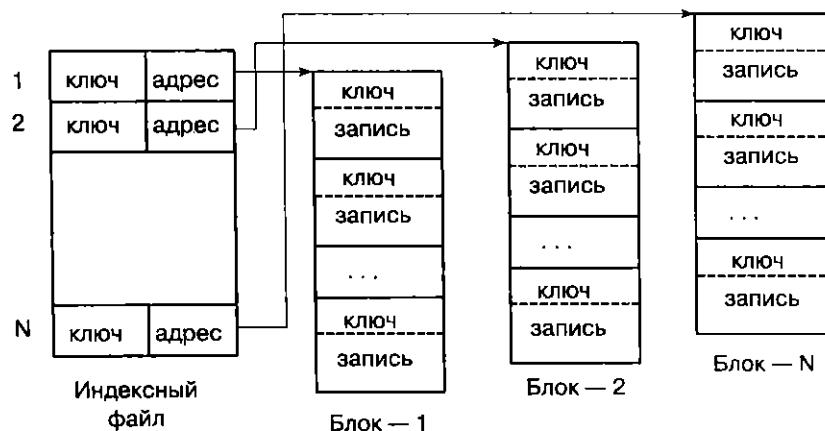


Рис. 3.3. Одноуровневая схема индексации

В каждом блоке записи располагаются в порядке возрастания значения ключа или свертки. Старшим ключом каждого блока является ключ его последней записи.

Если в индексном файле хранятся хеш-коды ключевых полей индексированной таблицы, то алгоритм поиска нужной записи (с указанным ключом) в таблице включает в себя следующие три этапа.

1. Образование свертки значения ключевого поля искомой записи.
2. Поиск в индексном файле записи о блоке, значение первого поля которого больше полученной свертки (это гарантирует нахождение искомой свертки в этом блоке).
3. Последовательный просмотр записей блока до совпадения сверток искомой записи и записи блока файла. В случае коллизий сверток ищется запись, значение ключа которой совпадает со значением ключа искомой записи.

Основным недостатком одноуровневой схемы является то, что ключи (свертки) записей хранятся вместе с записями. Это приводит к увеличению времени поиска записей из-за большой длины просмотра (значения данных в записях приходится пропускать).

Двухуровневая схема в ряде случаев оказывается более рациональной, в ней ключи (свертки) записей отделены от содержимого записей (рис. 3.4).

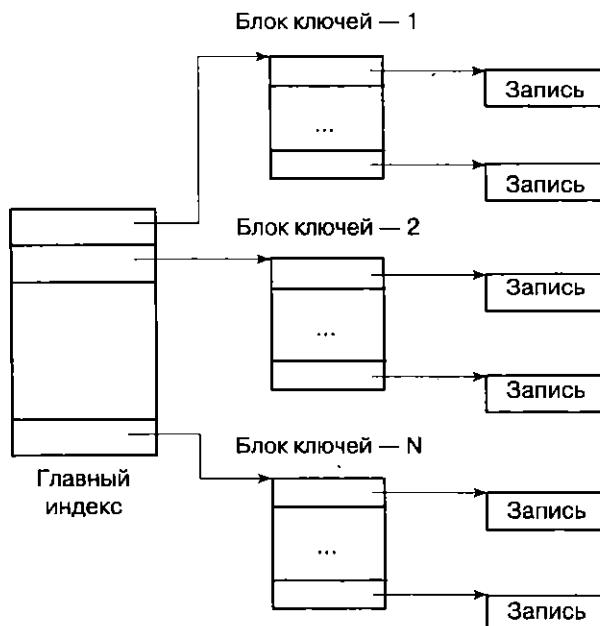


Рис. 3.4. Двухуровневая схема индексации

В этой схеме индекс основной таблицы распределен по совокупности файлов: одному файлу главного индекса и множеству файлов с блоками ключей.

На практике для создания индекса для некоторой таблицы БД пользователь указывает поле таблицы, которое требует индексации. Ключевые поля таблицы во многих СУБД как правило индексируются автоматически. Индексные файлы, создаваемые по ключевым полям таблицы, часто называются *файлами первичных индексов*.

Индексы, создаваемые пользователем для не ключевых полей, иногда называют *вторичными (пользовательскими) индексами*. Введение таких индексов не изменяет физического расположения записей таблицы, но влияет на последовательность просмотра записей. Индексные файлы, создаваемые для поддержания вторичных индексов таблицы, обычно называются *файлами вторичных индексов*.

Связь вторичного индекса с элементами данных базы может быть установлена различными способами. Один из них – использование вторичного индекса как входа для получения первичного ключа, по которому затем с использованием первичного индекса производится поиск необходимых записей (рис. 3.5).

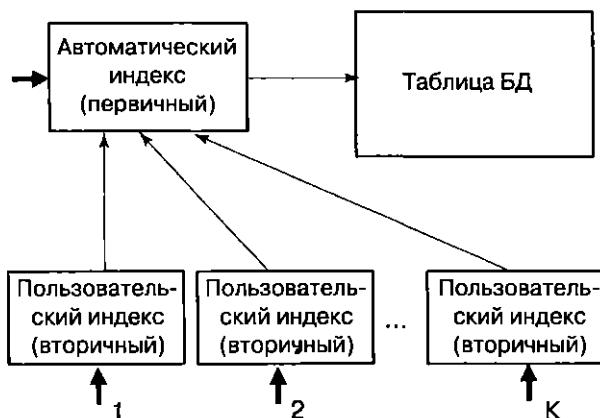


Рис. 3.5. Способ использования вторичных индексов

Некоторыми СУБД, например Access, деление индексов на первичные и вторичные не производится. В этом случае используются автоматически создаваемые индексы и индексы, определяемые пользователем по любому из не ключевых полей.

Главная причина повышения скорости выполнения различных операций в индексированных таблицах состоит в том, что основная часть работы про-

изводится с небольшими индексными файлами, а не с самими таблицами. Наиболеещий эффект повышения производительности работы с индексированными таблицами достигается для значительных по объему таблиц. Индексирование требует небольшого дополнительного места на диске и незначительных затрат процессора на изменение индексов в процессе работы. Индексы в общем случае могут изменяться перед выполнением запросов к БД, после выполнения запросов к БД, по специальным командам пользователя или программным вызовам приложений.

3.3. Связывание таблиц

При проектировании реальных БД информацию обычно размещают в нескольких таблицах. Таблицы при этом связаны семантикой информации. В реляционных СУБД для указания связей таблиц производят операцию их *связывания*.

Укажем выигрыш, обеспечиваемый в результате связывания таблиц. Многие СУБД при связывании таблиц автоматически выполняют контроль целостности вводимых в базу данных в соответствии с установленными связями. В конечном итоге это *повышает достоверность* хранимой в БД информации.

Кроме того, установление связи между таблицами *облегчает доступ* к данным. Связывание таблиц при выполнении таких операций, как поиск, просмотр, редактирование, выборка и подготовка отчетов, обычно обеспечивает возможность обращения к произвольным полям связанных записей. Это уменьшает количество явных обращений к таблицам данных и число манипуляций в каждой из них.

Основные виды связи таблиц

Между таблицами могут устанавливаться бинарные (между двумя таблицами), тернарные (между тремя таблицами) и, в общем случае, n-арные связи. Рассмотрим наиболее часто встречающиеся *бинарные связи*.

При связывании двух таблиц выделяют основную и дополнительную (подчиненную) таблицы. Логическое связывание таблиц производится с помощью *ключа связи*.

Ключ связи, по аналогии с обычным ключом таблицы, состоит из одного или нескольких полей, которые в данном случае называют *полями связи* (ПС).

Суть связывания состоит в установлении соответствия полей связи основной и дополнительной таблиц. Поля связи основной таблицы могут быть обычными и ключевыми. В качестве полей связи подчиненной таблицы чаще всего используют ключевые поля.

В зависимости от того, как определены поля связи основной и дополнительной таблиц (как соотносятся ключевые поля с полями связи), между двумя таблицами в общем случае могут устанавливаться следующие четыре основных вида связи (табл. 3.2):

- один – один (1:1);
- один – много (1:M);
- много – один (M:1);
- много – много (M:M или M:N).

Таблица 3.2

Характеристика видов связей таблиц

Характеристика полей связи по видам	1:1	1:M	M:1	M:M
Поля связи основной таблицы	являются ключом	являются ключом	не являются ключом	не являются ключом
Поля связи дополнительной таблицы	являются ключом	не являются ключом	являются ключом	не являются ключом

Дадим характеристику названным видам связи между двумя таблицами и приведем примеры их использования.

Связь вида 1:1

Связь вида 1:1 образуется в случае, когда все поля связи основной и дополнительной таблиц являются ключевыми. Поскольку значения в ключевых полях обеих таблиц не повторяются, обеспечивается взаимно-однозначное соответствие записей из этих таблиц. Сами таблицы, по сути, здесь становятся равноправными.

Пример 1.

Пусть имеются основная О1 и дополнительная Д1 таблицы. Ключевые поля обозначим символом «*», используемые для связи поля обозначим символом «+».

Таблица О1

* +

Поле11	Поле12
a	10
б	40
в	3

Таблица Д1

* +

Поле21	Поле22
a	стол
b	книга

В приведенных таблицах установлена связь между записью (а, 10) таблицы О1 и записью (а, стол) таблицы Д1. Основанием этого является совпадение значений в полях связи. Аналогичная связь существует и между записями (в, 3) и (в, книга) этих же таблиц. В таблицах записи отсортированы по значениям в ключевых полях.

Сопоставление записей двух таблиц по существу означает образование новых «виртуальных записей» (псевдозаписей). Так, первую пару записей логически можно считать новой псевдозаписью вида (а, 10, стол), а вторую пару — псевдозаписью вида (в, 3, книга).

На практике связи вида 1:1 используются сравнительно редко, так как хранимую в двух таблицах информацию легко объединить в одну таблицу, которая занимает гораздо меньше места в памяти ЭВМ. Возможны случаи, когда удобнее иметь не одну, а две и более таблицы. Причинами этого может быть необходимость ускорить обработку, повысить удобство работы нескольких пользователей с общей информацией, обеспечить более высокую степень защиты информации и т. д. Приведем пример, иллюстрирующий последнюю из приведенных причин.

Пример 2.

Пусть имеются сведения о выполняемых в некоторой организации научно-исследовательских работах. Эти данные включают в себя следующую информацию по каждой из работ: тему (девиз и полное наименование работы), шифр (код), даты начала и завершения работы, количество этапов, головного исполнителя и другую дополнительную информацию. Все работы имеют гриф «Для служебного пользования» или «секретно».

В такой ситуации всю информацию целесообразно хранить в двух таблицах: в одной из них — всю секретную информацию (например, шифр, полное наименование работы и головной исполнитель), а в другой — всю оставшуюся несекретную информацию. Обе таблицы можно связать по шифру работы. Первую из таблиц целесообразно защитить от несанкционированного доступа.

Связь вида 1:М

Связь 1:М имеет место в случае, когда одной записи основной таблицы соответствует несколько записей вспомогательной таблицы.

Пример 3.

Пусть имеются две связанные таблицы О2 и Д2. В таблице О2 содержится информация о видах мультимедиа-устройств ПЭВМ, а в таблице Д2 — сведения о фирмах-производителях этих устройств, а также о наличии на складе хотя бы одного устройства.

Таблица О2

Код	Вид устройства
а	CD-ROM
б	CD-Recorder
в	Sound Blaster

Таблица Д2

Код	Фирма-производитель	Наличие
а	Acer	да
а	Mitsumi	нет
а	NEC	да
а	Panasonic	да
а	Sony	да
б	Philips	нет
б	Sony	нет
б	Yamaha	да
в	Creative Labs	да

Таблица Д2 имеет два ключевых поля, так как одна и та же фирма может производить устройства различных видов. В примере фирма Sony производит устройства считывания и перезаписи с компакт-дисков.

Сопоставление записей обеих таблиц по полю «Код» порождает псевдозаписи вида: (а, CD-ROM, Acer, да), (а, CD-ROM, Mitsumi, нет), (а, CD-ROM, NEC, да), (а, CD-ROM, Panasonic, да), (а, CD-ROM, Sony, да), (б, CD-Recorder, Philips, нет), (б, CD-Recorder, Sony, да) и т. д.

Если свести псевдозаписи в новую таблицу, то получим полную информацию обо всех видах мультимедиа-устройств ПЭВМ, фирмах, их производящих, а также сведения о наличии конкретных видов устройств на складе.

Связь вида M:1

Связь M:1 имеет место в случае, когда одной или нескольким записям основной таблицы ставится в соответствие одна запись дополнительной таблицы.

Пример 4.

Рассмотрим связь таблиц ОЗ и ДЗ. В основной таблице ОЗ содержится информация о названиях деталей (Поле11), видах материалов, из которого детали можно изготовить (Поле12), и марках материала (Поле13). В дополнительной таблице ДЗ содержатся сведения о названиях деталей (Поле21), планируемых сроках изготовления (Поле22) и стоимости заказов (Поле23).

Таблица ОЗ

Поле11	Поле12	Поле13
деталь1	чугун	марка1
деталь1	чугун	марка2
деталь2	сталь	марка1
деталь2	сталь	марка2
деталь2	сталь	марка3
деталь3	алюминий	-
деталь4	чугун	марка2

Таблица ДЗ

Поле21	Поле22	Поле23
деталь1	4.03.98	90
деталь2	3.01.98	35
деталь3	17.02.98	90
деталь4	6.05.98	240

Связывание этих таблиц обеспечивает такое установление соответствия между записями, которое эквивалентно образованию следующих псевдозаписей: (деталь1, чугун, марка1, 4.03.98, 90), (деталь1, чугун, марка2, 4.03.98, 90), (деталь2, сталь, марка1, 3.01.98, 35), (деталь2, сталь, марка2, 3.01.98, 35), (деталь2, сталь, марка3, 3.01.98, 35), (деталь3, алюминий, —, 17.02.98, 90), (деталь4, чугун, марка2, 6.05.98, 240).

Полученная псевдотаблица может быть полезна при планировании или принятии управленческих решений, когда необходимо иметь все возможные вари-

анты исполнения заказов по каждому изделию. Отметим, что таблица О3 не имеет ключей и в ней возможно повторение записей. Если таблицу Д3 сделать основной, а таблицу О3 – дополнительной, получим связь вида 1:М. Поступив аналогично с таблицами О2 и Д2, можно получить связь вида М:1. Отсюда следует, что вид связи (1:М или М:1) зависит от того, какая таблица является главной, а какая дополнительной.

Связь вида М:М

Самый общий вид связи М:М возникает в случаях, когда некоторым записям основной таблицы соответствует несколько записей дополнительной таблицы.

Пример 5.

Пусть в основной таблице О4 содержится информация о том, на каких станках могут работать рабочие некоторой бригады. Таблица Д4 содержит сведения о том, кто из бригады ремонтников какие станки обслуживает.

Таблица О4

*	*	+
Работает	На станке	
Иванов А.В.	станок1	
Иванов А.В.	станок2	
Петров Н.Г.	станок1	
Петров Н.Г.	станок3	
Сидоров В.К.	станок2	

Таблица Д4

*	*	+
Обслуживает	Станок	
Голубев Б.С.	станок1	
Голубев Б.С.	станок3	
Зыков А.Ф.	станок2	
Зыков А.Ф.	станок3	

Первой и третьей записям таблицы О4 соответствует первая запись таблицы Д4 (у всех этих записей значение второго поля – «станок1»). Четвертой записи таблицы О4 соответствуют вторая и четвертая записи таблицы Д4 (во втором поле этих записей содержится «станок3»).

Исходя из определения полей связи этих таблиц можно составить новую таблицу с именем «О4+Д4», записями которой будут псевдозаписи. Записям полученной таблицы можно придать смысл возможных смен, составляемых при планировании работы. Для удобства, поля новой таблицы переименованы (кстати, такую операцию предлагают многие из современных СУБД).

Таблица «О4+Д4»

Работа	Станок	Обслуживание
Иванов А.В.	станок1	Голубев Б.С.
Иванов А.В.	станок2	Зыков А.Ф.
Петров Н.Г.	станок1	Голубев Б.С.
Петров Н.Г.	станок3	Голубев Б.С.
Петров Н.Г.	станок3	Зыков А.Ф.
Сидоров В.К.	станок2	Зыков А.Ф.

Приведенную таблицу можно использовать, например, для получения ответа на вопрос: «Кто обслуживает станки, на которых трудится Петров Н.Г.?».

Очевидно, аналогично связи 1:1, связь М:М не устанавливает подчиненности таблиц. Для проверки этого можно основную и дополнительную таблицу поменять местами и выполнить объединение информации путем связывания. Результирующие таблицы «О4+Д4» и «Д4+О4» будут отличаться порядком следования первого и третьего полей, а также порядком расположения записей.

Замечание.

На практике в связь обычно вовлекается сразу несколько таблиц. При этом одна из таблиц может иметь различного рода связи с несколькими таблицами. В случаях, когда связанные таблицы, в свою очередь, имеют связи с другими таблицами, образуется иерархия или дерево связей.

3.4. Контроль целостности связей

Из перечисленных видов связи чаще используется связь вида 1:М. Связь вида 1:1 можно считать частным случаем связи 1:М, когда одной записи главной таблицы соответствует одна запись вспомогательной таблицы. Связь М:1, по сути, является «зеркальным отображением» связи 1:М. Оставшийся вид связи М:М характеризуется как слабый вид связи или даже как отсутствие связи. Поэтому в дальнейшем рассматривается связь вида 1:М.

Напомним, что при образовании связи вида 1:М одна запись главной таблицы (главная, родительская запись) оказывается связанной с несколькими записями дополнительной (дополнительные, подчиненные записи) и имеет место схема, показанная на рис. 3.6.

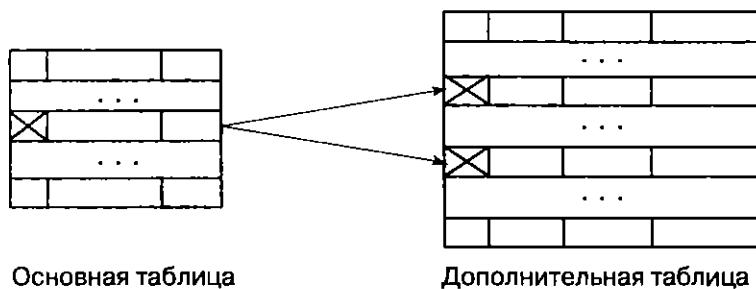


Рис. 3.6. Связь 1:М записей двух таблиц

Контроль целостности связей обычно означает анализ содержимого двух таблиц на соблюдение следующих правил:

- каждой записи основной таблицы соответствует нуль или более записей дополнительной таблицы;
- в дополнительной таблице нет записей, которые не имеют родительских записей в основной таблице;
- каждая запись дополнительной таблицы имеет только одну родительскую запись основной таблицы.

Опишем действие контроля целостности при манипулировании данными в таблицах. Рассмотрим три основные операции над данными двух таблиц:

- ввод новых записей,
- модификацию записей,
- удаление записей.

При рассмотрении попытаемся охватить все возможные методы организации контроля целостности. В реальных СУБД могут применяться собственные методы, подобные описываемым.

При **вводе новых записей** возникает вопрос определения последовательности ввода записей в таблицы такой, чтобы не допустить нарушение целостности. Исходя из приведенных правил, логичной является схема, при которой данные сначала вводятся в основную таблицу, а потом — в дополнительную. Очередность ввода может быть установлена на уровне целых таблиц или отдельных записей (случай одновременного ввода в несколько открытых таблиц).

В процессе заполнения основной таблицы контроль значений полей связи ведется как контроль обычного ключа (на совпадение со значениями тех же

полей других записей). Заполнение полей связи *дополнительной* таблицы контролируется на предмет совпадения со значениями полей связи основной таблицы. Если вновь вводимое значение в поле связи дополнительной таблицы не совпадет ни с одним соответствующим значением в записях основной таблицы, то ввод такого значения должен блокироваться.

Модификация записей. Изменение содержимого полей связанных записей, не относящихся к полям связи, очевидно, должно происходить обычным образом. Нас будет интересовать механизм изменения полей связи.

При редактировании полей связи дополнительной таблицы очевидным требованием является то, чтобы новое значение поля связи *совпадало* с соответствующим значением какой-либо записи основной таблицы. То есть дополнительная запись может сменить *родителя*, но остаться без него не должна.

Редактирование поля связи основной таблицы разумно подчинить одному из следующих правил:

- редактировать записи, у которых нет подчиненных записей. Если есть подчиненные записи, то блокировать модификацию полей связи;
- изменения в полях связи основной записи мгновенно передавать во все поля связи всех записей дополнительной таблицы (каскадное обновление).

В операциях *удаления записей* связанных таблиц большую свободу, очевидно, имеют записи дополнительной таблицы. Удаление их должно происходить практически бесконтрольно.

Удаление записей основной таблицы логично подчинить одному из следующих правил:

- удалять можно запись, которая не имеет подчиненных записей;
- запретить (блокировать) удаление записи при наличии подчиненных записей, либо удалять ее вместе со всеми подчиненными записями (каскадное удаление).

3.5. Теоретические языки запросов

Операции, выполняемые над отношениями, можно разделить на две группы. Первую группу составляют операции над множествами, к которым относятся операции: объединения, пересечения, разности, деления и декартова произведения. Вторую группу составляют специальные операции над отношениями, к которым, в частности, относятся операции: проекции, соединения, выбора.

В различных СУБД реализована некоторая часть операций над отношениями, определяющая в какой-то мере возможности данной СУБД и сложность реализации запросов к БД.

В реляционных СУБД для выполнения операций над отношениями используются две группы языков, имеющие в качестве своей *математической основы* теоретические языки запросов, предложенные Э.Коддом:

- реляционная алгебра;
- реляционное исчисление.

Эти языки представляют минимальные возможности реальных языков манипулирования данными в соответствии с реляционной моделью и эквивалентны друг другу по своим выразительным возможностям. Существуют не очень сложные правила преобразования запросов между ними.

В *реляционной алгебре* операнды и результаты всех действий являются отношениями. Языки реляционной алгебры являются *процедурными*, так как отношение, являющееся результатом запроса к реляционной БД, вычисляется при выполнении последовательности реляционных операторов, применяемых к отношениям. Операторы состоят из операндов, в роли которых выступают отношения, и реляционных операций. Результатом реляционной операции является отношение.

Языки *исчислений*, в отличие от реляционной алгебры, являются *непроцедурными* (описательными, или декларативными) и позволяют выражать запросы с помощью предиката первого порядка (высказывания в виде функций), которому должны удовлетворять кортежи или домены отношений. Запрос к БД, выполненный с использованием подобного языка, содержит лишь информацию о желаемом результате. Для этих языков характерно наличие наборов правил для записи запросов. В частности, к языкам этой группы относится SQL.

При рассмотрении языков реляционной алгебры и исчислений будем использовать базу данных, включающую в себя следующие таблицы:

- S (поставщики);
- P (детали);
- SP (поставки).

Первичными ключами этих таблиц являются соответственно: П# (код поставщика), Д# (код детали) и составной ключ (П#, Д#). Содержимое таблиц приведено на рис. 3.7. Для удобства изложения предположим, что в рассматриваемых языках запросов нет ограничений на употребление символов русского алфавита в именах атрибутов. Каждое из полей П# и Д# таблицы SP в отдельности является внешним ключом по отношению к таблице S и P соответственно.

Предположим, что имена доменов (множеств допустимых значений) совпадают с именами атрибутов. Исключением составляют атрибуты Город_П (город, в котором находится поставщик) и Город_Д (город, в котором выпускается деталь), которые имеют общий домен: множество названий городов. Имя этого домена может быть, например, просто Город. Характеристики доменов как типов данных следующие: Д# — строка символов длиной 5, Имя — строка символов длиной 20, Статус — цифровое длиной 5, Город — строка симво-

S

П#	Имя	Статус	Город_П
S1	Сергей	20	Москва
S2	Иван	10	Киев
S3	Борис	30	Киев
S4	Николай	20	Москва
S5	Андрей	30	Минск

P

Д#	Название	Тип	Вес	Город_Д
P1	гайка	каленый	12	Москва
P2	болт	мягкий	17	Киев
P3	винт	твёрдый	17	Ростов
P4	винт	каленый	14	Москва
P5	палец	твёрдый	12	Киев
P6	шпилька	каленый	19	Москва

SP

П#	Д#	Количество
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Рис. 3.7. Таблицы поставщиков, деталей и поставок

лов длиной 15, Д# — строка символов длиной 6, Тип — строка символов длиной 6, Вес — цифровое длиной 5, Количество — цифровое длиной 5.

3.6. Реляционная алгебра

Реляционная алгебра как теоретический язык запросов по сравнению с реляционным исчислением более наглядно описывает выполняемые над отношениями действия.

Примером языка запросов, основанного на реляционной алгебре, является ISBL (Information System Base Language — базовый язык информационных систем). Языки запросов, построенные на основе реляционной алгебры, в современных СУБД широкого распространения не получили. Однако знакомство с ней полезно для понимания сути реляционных операций, выражаемых другими используемыми языками.

Вариант реляционной алгебры, предложенный Коддом, включает в себя следующие *основные операции*: объединение, разность (вычитание), пересечение, декартово (прямое) произведение (или произведение), выборка (селекция, ограничение), проскция, деленис и соединение. Упрощенное графическое представление этих операций приведено на рис. 3.8.

По справедливому замечанию Дайта, реляционная алгебра Кодда обладает некоторыми недостатками. Во-первых, восемь перечисленных операций по охвату своих функций, с одной стороны, избыточны, так как минимально необходимый набор составляют пять операций: объединение, вычитание, произведение, проекция и выборка. Три другие операции (пересечение, соединение и деление) можно определить через пять минимально необходимых. Так, например, соединение — это проекция выборки произведения.

Во-вторых, этих восьми операций недостаточно для построения реальной СУБД на принципах реляционной алгебры. Требуются расширения, включающие операции: переименования атрибутов, образования новых вычисляемых атрибутов, вычисления итоговых функций, построения сложных алгебраических выражений, присвоения, сравнения и т. д.

Рассмотрим перечисленные операции более подробно, сначала — операции реляционной алгебры Кодда, а затем — дополнительные операции, введенные Дайтом.

Операции реляционной алгебры Кодда можно разделить на *две* группы: *базовые теоретико-множественные* и *специальные реляционные*. Первая группа операций включает в себя классические операции теории множеств: объединение, разность, пересечение и произведение. Вторая группа представляет собой развитие обычных теоретико-множественных операций в направлении к реальным задачам манипулирования данными, в ее состав входят следующие операции: проекция, селекция, деление и соединение.

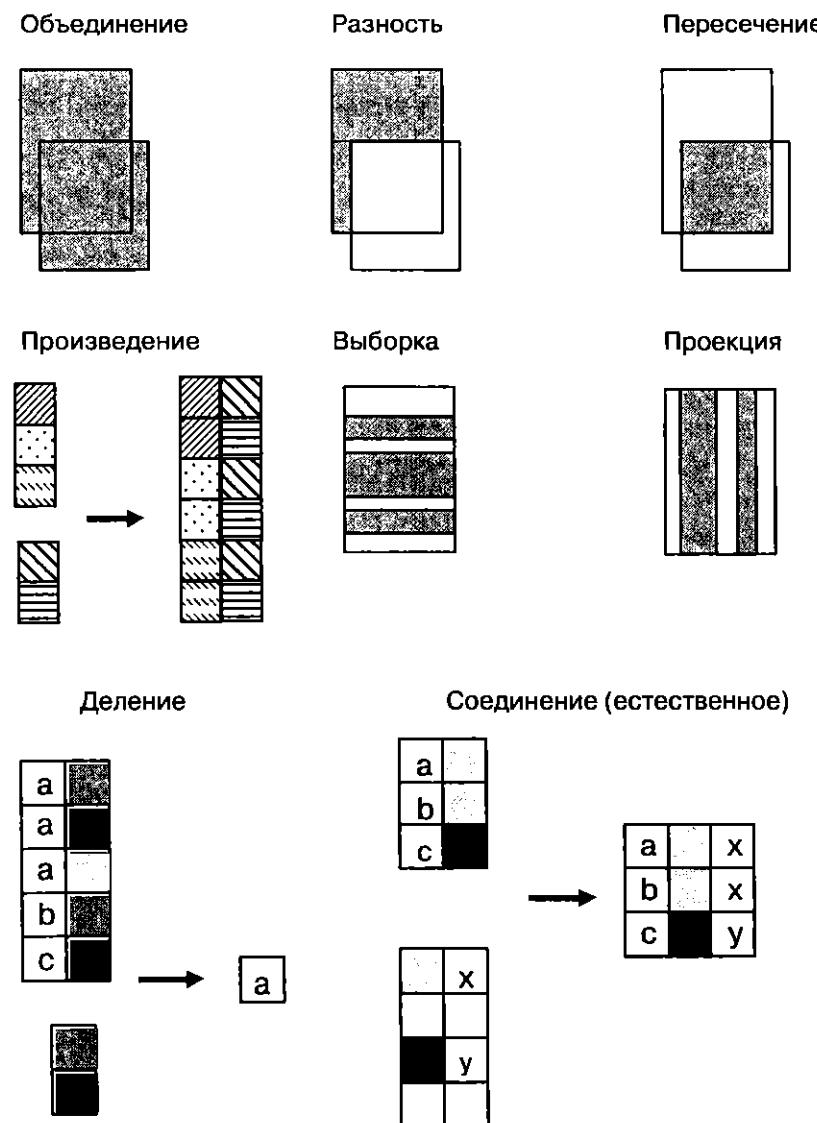


Рис. 3.8. Основные операции реляционной алгебры

Операции реляционной алгебры могут выполняться над одним отношением (например, проекция) или над двумя отношениями (например, объединение). В первом случае операция называется унарной, а во втором — бинарной. При выполнении бинарной операции участвующие в операциях отношения должны быть совместимы по структуре.

Совместимость структур отношений означает совместимость имен атрибутов и типов соответствующих доменов. Частным случаем совместимости является идентичность (совпадение). Для устранения конфликтов имен атрибутов в исходных отношениях (когда совпадение имен недопустимо), а также для построения произвольных имен атрибутов результирующего отношения применяется операция переименования атрибутов. Структура результирующего отношения по определенным правилам наследует свойства структур исходных отношений. В большинстве рассматриваемых бинарных реляционных операций будем считать, что заголовки исходных отношений идентичны, так как в этом случае не возникает проблем с заголовком результирующего отношения (в общем случае, заголовки могут не совпадать, тогда нужно оговаривать правила формирования заголовка отношения-результата).

Объединением двух совместимых отношений R1 и R2 одинаковой размерности ($R1 \text{ UNION } R2$) является отношение R, содержащее все элементы исходных отношений (с исключением повторений).

Пример 1. Объединение отношений.

Пусть отношение обозначает R1 множество поставщиков из Лондона, а отношение R2 — множество поставщиков, которые поставляют деталь P1. Тогда отношение R обозначает поставщиков, находящихся в Лондоне, или поставщиков, выпускающих деталь P1, либо тех и других.

R1

П#	Имя	Статус	Город_П
S1	Сергей	20	Москва
S4	Николай	20	Москва

R2

П#	Имя	Статус	Город_П
S1	Сергей	20	Москва
S2	Иван	10	Киев

R ($R1 \text{ UNION } R2$)

П#	Имя	Статус	Город_П
S1	Сергей	20	Москва
S2	Иван	10	Киев
S4	Николай	20	Москва

Вычитание совместимых отношений R1 и R2 одинаковой размерности ($R1 \text{ MINUS } R2$) есть отношение, тело которого состоит из множества кортежей,

принадлежащих R1, но не принадлежащих отношению R2. Для тех же отношений R1 и R2 из предыдущего примера отношение R будет представлять собой множество поставщиков, находящихся в Лондоне, но не выпускающих деталь P1, то есть $R = \{(S4, Николай, 20, Москва)\}$.

Заметим, что результат операции вычитания зависит от порядка следования операндов, то есть $R1 MINUS R2$ и $R2 MINUS R1$ — не одно и то же.

Пересечение двух совместимых отношений R1 и R2 одинаковой размерности ($R1 INTERSECT R2$) порождает отношение R с телом, включающим в себя кортежи, одновременно принадлежащие обоим исходным отношениям. Для отношений R1 и R2 результирующее отношение R будет означать всех производителей из Лондона, выпускающих деталь P1. Тело отношения R состоит из единственного элемента ($S1, Сергей, 20, Москва$).

Произведение отношения R1 степени $k1$ и отношения R2 степени $k2$ ($R1 TIMES R2$), которые не имеют одинаковых имен атрибутов, есть такое отношение R степени $(k1+k2)$, заголовок которого представляет сцепление заголовков отношений R1 и R2, а тело имеет кортежи такие, что первые $k1$ элементов кортежей принадлежат множеству R1, а последние $k2$ элементов — множеству R2. При необходимости получить произведение двух отношений, имеющих одинаковые имена одного или нескольких атрибутов, применяется операция переименования RENAME, рассматриваемая далее.

Пример 2. Произведение отношений.

Пусть отношение R1 представляет собой множество номеров всех текущих поставщиков {S1, S2, S3, S4, S5}, а отношение R2 — множество номеров всех текущих деталей {P1, P2, P3, P4, P5, P6}. Результатом операции $R1 TIMES R2$ является множество всех пар типа «поставщик — деталь», то есть $\{(S1, P1), (S1, P2), (S1, P3), (S1, P4), (S1, P5), (S1, P6), (S2, P1), \dots, (S5, P6)\}$.

Заметим, что в теории множеств результатом операции прямого произведения является множество, каждый элемент которого является парой элементов, первый из которых принадлежит R1, а второй — принадлежит R2. Поэтому кортежами декартова произведения бинарных отношений будут кортежи вида: $((a, b), (v, g))$, где кортеж (a, b) принадлежит отношению R1, а кортеж (v, g) — принадлежит отношению R2. В реляционной алгебре применяется расширенный вариант прямого произведения, при котором элементы кортежей двух исходных отношений сливаются, что при записи кортежей результирующего отношения означает удаление лишних скобок, то есть (a, b, v, g) .

Выборка ($R WHERE f$) отношения R по формуле f представляет собой новое отношение с таким же заголовком и телом, состоящим из таких кортежей отношения R, которые удовлетворяют истинности логического выражения, заданного формулой f . Для записи формулы используются операнды — имена атрибутов (или номера столбцов), константы, логические операции (AND — И, OR — ИЛИ, NOT — НЕ), операции сравнения и скобки.

Примеры 3. Выборки.

P WHERE Вес < 14

Д#	Название	Тип	Вес	Город_Д
P1	гайка	каленый	12	Москва
P5	палец	твёрдый	12	Киев

SP WHERE П# = "S1" AND Д# = "P1"

П#	Д#	Количество
S1	P1	300

Проекция отношения A на атрибуты X, Y, ..., Z (A [X, Y, ..., Z]), где множество {X, Y, ..., Z} является подмножеством полного списка атрибутов заголовка отношения A, представляет собой отношение с заголовком X, Y, ..., Z и телом, содержащим кортежи отношения A, за исключением повторяющихся кортежей. Повторение одинаковых атрибутов в списке X, Y, ..., Z запрещается.

- Операция проекции допускает следующие дополнительные варианты записи:
- отсутствие списка атрибутов подразумевает указание всех атрибутов (операция тождественной проекции);
 - выражение вида R[] означает *пустую* проекцию, результатом которой является пустое множество;
 - операция проекции может применяться к произвольному отношению, в том числе и к результату выборки.

Примеры 4. Проекции.

P [Тип, Город_Д]

(S WHERE Город_П = "Киев") [П#]

Тип	Город_Д
каленый	Москва
мягкий	Киев
твёрдый	Ростов
твёрдый	Киев

П#	Город_П
S2	Киев
S3	Киев

Результатом **деления** отношения R1 с атрибутами A и B на отношение R2 с атрибутом B (R1 DIVIDE BY R2), где A и B простые или составные атрибуты, причем атрибут B – общий атрибут, определенный на одном и том же домене (множестве доменов составного атрибута), является отношение R с заголовком A и телом, состоящим из кортежей г таких, что в отношении R1 имеются кортежи (г, s), причем множество значений s включает множество значений атрибута B в отношении R2.

Пример 5. Деление отношения.

Пусть $R1$ — проекция $SP[\Pi\#, D\#]$, а $R2$ — отношение с заголовком $D\#$ и телом $\{P2, P4\}$, тогда результатом деления $R1$ на $R2$ будет отношение R с заголовком $\Pi\#$ и телом $\{S1, S4\}$.

R1	
$\Pi\#$	$D\#$
S1	P1
S1	P2
S1	P3
S1	P4
S1	P5
S1	P6
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4
S4	P5

R2	
$D\#$	
P2	
P4	

R1 DIVIDE BY R2	
$\Pi\#$	
S1	
S4	

Соединение $C_f(R1, R2)$ отношений $R1$ и $R2$ по условию, заданному формулой f , представляет собой отношение R , которое можно получить путем Декартова произведения отношений $R1$ и $R2$ с последующим применением к результату операции выборки по формуле f . Правила записи формулы f такие же, как и для операции селекции.

Другими словами, соединением отношения $R1$ по атрибуту А с отношением $R2$ по атрибуту В (отношения не имеют общих имен атрибутов) является результат выполнения операции вида:

$(R1 \text{ TIMES } R2) \text{ WHERE } A \Theta B,$

где Q — логическое выражение над атрибутами, определенными на одном (нескольких — для составного атрибута) домене. Соединение $C_f(R1, R2)$, где формула f имеет произвольный вид (в отличие от частных случаев, рассматриваемых далее), называют также *Q-соединением*.

Важными с практической точки зрения частными случаями соединения являются эквисоединение и естественное соединение.

Операция **эквисоединения** характеризуется тем, что формула задает равенство операндов. Приведенный выше пример демонстрирует частный случай операции эквисоединения по одному столбцу. Иногда эквисоединение двух отно-

шений выполняется по таким столбцам, атрибуты которых в обоих отношениях имеют соответственно одинаковые имена и домены. В этом случае говорят об эквисоединении по общему атрибуту.

Операция *естественного соединения* (операция JOIN) применяется к двум отношениям, имеющим общий атрибут (простой или составной). Этот атрибут в отношениях имеет одно и то же имя (совокупность имен) и определен на одном и том же домене (доменах).

Результатом операции естественного соединения является отношение R, которое представляет собой проекцию эквисоединения отношений R1 и R2 по общему атрибуту на объединенную совокупность атрибутов обоих отношений.

Пример 6. Q-соединение.

Необходимо найти соединение отношений S и P по атрибутам Город_П и Город_Д соответственно, причем кортежи результирующего отношения должны удовлетворять отношению «больше» (в смысле лексикографического порядка — по алфавиту).

(S TIMES P) WHERE Город_П > Город_Д

Пример 7. Эквисоединение.

П#	Имя	Статус	Город_П	Д#	Название	Тип	Вес	Город_Д
S2	Иван	10	Киев	P1	гайка	каленый	12	Москва
S2	Иван	10	Киев	P4	винт	каленый	14	Москва
S2	Иван	10	Киев	P6	шпилька	каленый	19	Москва
S3	Борис	30	Киев	P1	гайка	каленый	12	Москва
S3	Борис	30	Киев	P4	винт	каленый	14	Москва
S3	Борис	30	Киев	P6	шпилька	каленый	19	Москва

Пусть необходимо найти естественное соединение отношений S и P по общему атрибуту, характеризующему город (в отношении S — это Город_П, а в отношении P — Город_Д). Поскольку условие операции требует одинакости имен атрибутов, по которым выполняется соединение, то применяется операция RENAME переименования атрибутов.

(S RENAME Город_П AS Город) JOIN (P RENAME Город_Д AS Город)

П#	Имя	Статус	Город	Д#	Название	Тип	Вес
S1	Сергей	20	Москва	P1	гайка	каленый	12
S1	Сергей	20	Москва	P4	винт	каленый	14
S1	Сергей	20	Москва	P6	шпилька	каленый	19
S2	Иван	10	Киев	P2	болт	мягкий	17
S2	Иван	10	Киев	P5	палец	твёрдый	12

S3	Борис	30	Киев	P2	болт	мягкий	17
S3	Борис	30	Киев	P5	палец	твёрдый	12
S4	Николай	20	Москва	P1	гайка	каленый	12
S4	Николай	20	Москва	P4	винт	каленый	14
S4	Николай	20	Москва	P6	шпилька	каленый	19

Дополнительные операции реляционной алгебры, предложенные Дейтом, включают следующие операции.

Операция **переименования** позволяет изменить имя атрибута отношения и имеет вид:

RENAME <исходное отношение> <старое имя атрибута> AS <новое имя атрибута>,

где <исходное отношение> задается именем отношения либо выражением реляционной алгебры. В последнем случае выражение заключают в круглые скобки.

Например, RENAME Город_П AS Город_размещения_Поставщика.

Замечание.

Для удобства записи выражений одновременного переименования нескольких атрибутов Дейтом введена операция **множественного переименования**, которая представима следующим образом:

RENAME <отн.> <ст. имя атр.1> AS <нов. имя атр.1>,<ст. имя атр.2> AS <нов. имя атр.2>, ..., <ст. имя атр.N> AS <нов. имя атр.N>.

Операция **расширения** порождает новое отношение, похожее на исходное, но отличающееся наличием добавленного атрибута, значения которого получаются путем некоторых скалярных вычислений. Операция расширения имеет вид:

EXTEND <исходное отношение> ADD <выражение> AS <новый атрибут>,

где к исходному отношению добавляется (ключевое слово ADD) <новый атрибут>, подсчитываемый по правилам, заданным <выражением>. Исходное отношение может быть задано именем отношения и с помощью выражения реляционной алгебры, заключенного в круглые скобки. При этом имя нового атрибута не должно входить в заголовок исходного отношения и не может использоваться в <выражении>. Помимо обычных арифметических операций и операций сравнения, в выражении можно использовать различные функции, называемые итоговыми, такие как: COUNT (количество), SUM (сумма), AVG (среднее), MAX (максимальное), MIN (минимальное).

Например,

EXTEND (P JOIN SP) ADD (Вес * Количество) AS Общий_Вес.

Замечания.

- Пользуясь операцией расширения, можно выполнить переименование атрибута. Для этого нужно в выражении указать имя атрибута, в конструкции AS определить новое имя этого атрибута, затем выполнить проекцию полученного отношения на множество атрибутов, исключая старый атрибут. Таким образом, запись (EXTEND S ADD Город_II AS Город) [П#, Имя, Статус, Город] эквивалента конструкции S RENAME Город_II AS Город.
- Подобно тому, как это сделано для операции *переименования*, Дейт определил операцию *множественного расширения*, которая позволяет в одной синтаксической конструкции вычислять несколько новых атрибутов. Формально операция представима следующим образом:

EXTEND <отн.> ADD <выр.1> AS <атр.1>, <выр.2>
AS <атр.2>, ..., <выр.N> AS <атр.N>.

Операция **подведения итогов** SUMMARIZE выполняет «вертикальные» или групповые вычисления и имеет следующий формат:

SUMMARIZE <исх. отн.> BY (<список атрибутов>) ADD <выр.>
AS <новый атрибут>,

где исходное отношение задается именем отношения либо заключенным в круглые скобки выражением реляционной алгебры, <список атрибутов> представляет собой разделенные запятыми имена атрибутов исходного отношения A1, A2, ..., AN, <выр.> — скалярное выражение, аналогичное выражению операции EXTEND, а <новый атрибут> — имя формируемого атрибута. В списке атрибутов и в выражении не должен использоваться <новый атрибут>.

Результатом операции SUMMARIZE является отношение R с заголовком, состоящим из атрибутов списка, расширенного новым атрибутом. Для получения тела отношения R сначала выполняется проецирование (назовем проекцию R1) исходного отношения на атрибуты A1, A2, ..., AN, после чего каждый кортеж проекции расширяется новым (N+1)-м атрибутом. Поскольку проецирование, как правило, приводит к сокращению количества кортежей по отношению к исходному отношению (удаляются одинаковые кортежи), то можно считать, что происходит своеобразное группирование кортежей исходного отношения: одному кортежу отношения R1 соответствует один или более (если было дублирование при проецировании) кортежей исходного отношения. Значение (N+1)-го атрибута каждого кортежа отношения R формируется путем вычисления выражения над соответствующей этому кортежу группой кортежей исходного отношения.

Пример 8. Подведение итогов.

Пусть требуется вычислить количество поставок по каждому из поставщиков.

SUMMARIZE SP BY (П#) ADD COUNT AS Количество_поставок

П#	Количество_поставок
S1	6
S2	2
S3	1
S4	3

Отметим, что функция COUNT определяет количество кортежей в каждой из групп исходного отношения.

Операция *множественного подведения итогов*, подобно соответствующим операциям переименования и расширения, выполняет одновременно несколько «вертикальных» вычислений и записывает результаты в отдельные новые атрибуты. Простейшим примером такой операции может служить следующая запись:

SUMMARIZE SP BY (Д#) ADD SUM Количество AS Общее_число_поставок AVG Количество AS Среднее_число_поставок.

К основным операторам, позволяющим изменять тело существующего отношения, отнесем операции реляционного *присвоения, вставки, обновления и удаления*.

Операцию *присвоения* можно представить следующим образом:

<выражение-цель> := <выражение-источник>,

где оба выражения задают совместимые (точнее, эквивалентные) по структуре отношения. Типичный случай выражений: в левой части — имя отношения, а в правой — некоторое выражение реляционной алгебры. Выполнение операции присвоения сводится к замене предыдущего значения отношения на новое (начальное значение, если тело отношения было пустым), определенное выражением-источником.

С помощью операции присвоения можно не только полностью заменить все значения отношения-цели, но и добавить или удалить кортежи. Приведем пример, в котором в отношение S дописывается один кортеж:

S := S UNION {{< П# : 'S6' >, < Имя : 'Борис' >, < Статус : '50' >, < Город_П : 'Мадрид' >}}.

Более удобными операциями изменения тела отношения являются операции *вставки, обновления и удаления*.

Операция *вставки* INSERT имеет следующий вид:

INSERT <выражение-источник> INTO <выражение-цель>,

где оба выражения должны быть совместимы по структуре. Выполнение операции сводится к вычислению <выражение-источник> и вставке полученных кортежей в отношение, заданное <выражение-цель>.

Пример. `INSERT (S WHERE Город_П='Москва') INTO Темп.`

Операция **обновления** `UPDATE` имеет следующий вид:

`UPDATE <выражение-цель> <список элементов>,`

где `<список элементов>` представляет собой последовательность разделенных запятыми операций присвоения `<атрибут> := <скалярное выражение>`. Результатом выполнения операции обновления является отношение, полученное после присвоения соответствующих значений атрибутам отношения, заданного целевым выражением.

Например, `UPDATE P WHERE Тип='каленый' Город := 'Киев'`. Эта операция предписывает изменить значение атрибута Город (независимо от того, каким оно было) на новое значение — 'Киев' таких кортежей отношения P, атрибут Тип которых имеет значение 'каленый'.

Операция **удаления** `DELETE` имеет следующий вид:

`DELETE <выражение-цель>,`

где `<выражение-цель>` представляет собой реляционное выражение, описывающее удаляемые кортежи.

Например, `DELETE S WHERE Статус < 20`.

Операция **реляционного сравнения** может использоваться для прямого сравнения двух отношений. Она имеет синтаксис:

`<выражение1> Θ <выражение2>,`

где оба выражения задают совместимые по структуре отношения, а знак Θ — один из следующих операторов сравнения: = (равно), \neq (не равно), \leq (собственное подмножество), $<$ (подмножество), \geq (надмножество), $>$ (собственное надмножество).

Например, сравнение: «Совпадают ли города поставщиков и города хранения деталей?» можно записать так: `S [Город_П] = P [Город_Д]`.

Основные правила записи выражений. Как отмечалось, результатом произвольной реляционной операции является отношение, которое, в свою очередь, может участвовать в другой реляционной операции. Это свойство реляционной алгебры называется *свойством замкнутости*.

Свойство замкнутости позволяет записывать *вложенные выражения* реляционной алгебры, основой которых выступают рассмотренные ранее элементарные операции: объединение, проекция, пересечение, выборка и т. д.

При записи произвольного выражения реляционной алгебры надо принимать во внимание следующее.

1. В реляционной алгебре должен быть определен приоритет выполнения операций (например, операция пересечение более приоритетна чем операция объединение), который нужно учитывать при записи выражений.

Для изменения порядка выполнения операций в выражениях можно использовать круглые скобки.

2. Существуют тождественные преобразования, позволяющие по-разному записывать одно и то же выражение. Например, следующие выражения эквивалентны (здесь А – отношение, С, С1, С2 – выражения):

A WHERE C1 AND C2 и (A WHERE C1) INTERSECT (A WHERE C2),
A WHERE C1 OR C2 и (A WHERE C1) UNION (A WHERE C2),
A WHERE NOT C и A MINUS (A WHERE C).

3. Составляя выражение, нужно обеспечивать совместимость участвующих в операциях отношений. При необходимости изменения заголовков следует выполнять переименование атрибутов.

3.7. Реляционное исчисление

Как отмечалось ранее, принципиальное различие между реляционной алгеброй и реляционным исчислением состоит в том, что в первом случае процесс получения искомого результата описывается явным образом путем указания набора операций, которые надо выполнить для получения результата, а во втором – указываются свойства искомого отношения без конкретизации процедуры его получения.

Внешне подходы сильно отличаются: один из них предписывающий (реляционная алгебра), а другой описательный (реляционное исчисление). На более низком уровне рассмотрения подходы эквивалентны, так как любые выражения реляционной алгебры могут быть преобразованы в семантически эквивалентные выражения реляционного исчисления и наоборот. Возможность такого преобразования доказывалась многими авторами, в частности, для этого можно использовать *алгоритм редукции Кодда*.

Для названного алгоритма преобразования покажем на содержательном уровне возможности формулировки одного и того же запроса с помощью реляционной алгебры и реляционного исчисления на простом примере.

Пусть запрос выглядит следующим образом: «Получить номера и города поставщиков, выпускающих деталь Р2».

Словесно алгебраическая версия этого запроса описывается так:

- образовать естественное соединение отношений S и SP по атрибуту П#;
- выбрать из результата этого соединения кортежи с деталью Р2 (в поле Д# должна быть строка Р2);
- спроектировать результат предыдущей операции на атрибуты П# и Город_П.

Этот же запрос в терминах реляционного исчисления можно сформулировать примерно так: «Получить атрибуты П# и Город_П для таких поставщиков, для которых существует поставка в отношении SP с тем же значением атрибута П# и со значением Р" атрибута Д#».

Результатом выполнения запроса будет отношение R вида:

П#	Город_П
S1	Москва
S2	Киев
S3	Киев
S4	Москва

Читатель может самостоятельно убедиться в этом, проделав описанные выше операции реляционной алгебры.

Преимуществом реляционного исчисления перед реляционной алгеброй можно считать то, что пользователю не требуется самому строить алгоритм выполнения запроса. Программа СУБД (при достаточной ее интеллектуальности) сама строит эффективный алгоритм.

Отметим, что поставленную задачу выборки можно решить более оптимально с точки зрения потребности в оперативной памяти. Более экономичный вариант решения в терминах операций реляционной алгебры выглядит так:

- выбрать из отношения SP кортежи, относящиеся к детали P2;
- выполнить естественное соединение отношения S и отношения, полученного на предыдущем шаге;
- спроектировать текущее отношение на атрибуты П# и Город_П.

Экономия памяти при реализации этого алгоритма в сравнении с первоначальным вариантом достигается за счет снижения размерности участвующих в операциях временных таблиц, необходимых для хранения промежуточных результатов. Если в предыдущем случае размерность временной таблицы была $12*6$ (12 строк на 6 колонок), то в последнем случае — $4*6$.

Математической основой реляционного исчисления является исчисление предикатов — один из разделов математической логики. Понятие реляционного исчисления как языка работы с базами данных впервые предложено Коддом. Им же был разработан язык ALPHA — прототип программно реализованного языка QUEL, который некоторое время конкурировал с языком SQL.

Существует два варианта исчислений: *исчисление кортежей* и *исчисление доменов*. В первом случае для описания отношений используются переменные, допустимыми значениями которых являются кортежи отношения, а во втором случае — элементы домена.

Реляционное исчисление, основанное на кортежах (*исчисление кортежей*), предложено и реализовано при разработке упоминавшегося языка ALPHA. В нем, как и в процедурных языках программирования, сначала нужно описать используемые переменные, а затем записывать некоторые выражения.

Описательную часть исчисления можно представить в виде:

RANGE OF <переменная> IS <список>,

где прописными буквами записаны ключевые слова языка, <переменная> – идентификатор переменной кортежа (области значений), а <список> – последовательность одного или более элементов, разделенных запятыми, то есть конструкция вида: $x_1, [x_2, \dots, x_n] \dots$.

Вся конструкция RANGE указывает идентификатор переменной и область ее допустимых значений. Список элементов $x_1, [x_2, \dots, x_p] \dots$ содержит элементы, каждый из которых является либо отношением, либо выражением над отношением (порядок записи выражений описывается далее). Все элементы списка должны быть совместимы по типу, то есть соответствующие элементам отношения должны иметь идентичные заголовки. Область допустимых значений <переменной> образуется путем объединения значений всех элементов списка. Так, запись вида RANGE OF T IS X1,X2 означает, что область определения переменной T включает в себя все значения из отношения, которое является объединением отношений X1 и X2.

Пример 1. Варианты описаний.

RANGE OF SX IS S;

RANGE OF SPX IS SP;

RANGE OF SY IS (SX) WHERE SX.Город_П = 'Москва',

(SX) WHERE EXISTS SPX (SPX.П# = SX.П# AND
SPX.Д# = 'P1');

Переменные SX и SPX в первых двух примерах определены соответственно на отношениях S и SP соответственно (рис. 3.7). Третий пример иллюстрирует запись выражений при определении переменной кортежа. Переменная SY здесь может принимать значения из множества кортежей отношения S для поставщиков из Лондона или поставщиков, которые поставляют деталь P1 (или для тех и других).

Синтаксис выражений исчисления рассматривается ниже. Запись выражения, формулирующего запрос на языке исчисления кортежей с помощью формы Бэкуса-Наура, упрощенно можно представить следующим образом:

```

<выражение> ::= (y1, [y2, ..., ym]...) [ WHERE wff ]
y ::= {<переменная> | <переменная>.〈атрибут〉} [ AS <атрибут> ]
wff ::= <условие> |
      NOT wff |
      <условие> AND wff |
      <условие> OR wff |
      IF <условие> THEN wff |
      EXISTS <переменная> (wff) |
      FORALL <переменная> (wff) |
      (wff)
  
```

Общий смысл записи выражения состоит в перечислении атрибутов результирующего (целевого) отношения, атрибуты которого должны удовлетворять условию истинности формулы wff (well formulated formula – правильно построенная формула). Список атрибутов целевого отношения, или *целевой список*, в терминах реляционной алгебры по существу определяет операцию проекции, а формула wff – селекцию кортежей.

В паре <переменная>.<атрибут> первая составляющая служит для указания переменной кортежа (определенной конструкцией RANGE), а вторая – для определения атрибута отношения, на котором изменяется переменная кортежа. Необязательная часть «AS <атрибут>» используется для переименования целевого отношения. Если она отсутствует, то имя атрибута целевого отношения наследуется от соответствующего имени атрибута исходного отношения.

Употребление в качестве элемента целевого отношения просто имени переменной Т равносильно перечислению в списке всех соответствующих атрибутов, т. е. Т.А₁, Т.А₂, …, Т.А_n, где А₁, А₂, …, А_n – атрибуты отношения, сопоставляемого с переменной Т.

Пример 2. Варианты записи пары <переменная>.<атрибут>.

SX.П#

SX.П# AS Город_Поставщика

SX

SX.П#, SX.Город_П AS Город_Поставщика, РХ.Д#, РХ.Город_Д# AS
Город_Детали

В приведенном определении wff <условие> представляет собой либо формулу wff, заключенную в скобки, либо простое сравнение вида:

<операнд1> Θ <операнд2>,

где в качестве любого операнда выступает переменная или скалярная константа, а символ Θ обозначает операцию сравнения =, ≠, >, ≥, <, ≤ и т. д.

Ключевые слова NOT, AND и OR обозначают логические операции соответственно: И, НЕ и ИЛИ. Ключевые слова IF и THEN переводятся соответственно «если» и «то». И наконец, ключевые слова EXISTS и FORALL называются **кванторами**. Первый из них – квантор существования, а второй – квантор всеобщности. Рассмотрим эти кванторы несколько подробнее.

Формула wff вида: EXISTS x (f) означает: «Существует по крайней мере одно такое значение переменной x, что вычисление формулы f дает значение истина». Выражение вида: FORALL x (f) интерпретируется как высказывание: «Для всех значений переменной x вычисление формулы f дает значение истина». В общем случае переменные кортежей в формулах могут быть *свободными* или *связанными*. В формулах EXISTS x (f) и FORALL x (f) переменные кортежей x всегда являются связанными.

Пример 3. Запись выражения.

Приведем запись выражения, соответствующее запросу: «Получить имена поставщиков, которые поставляют все детали».

$SX.\text{Город_П} \text{ WHERE FORALL } PX (\text{EXISTS SPX}$

$$(\text{SPX.П\#} = SX.\text{П\#} \text{ AND} \\ \text{SPX.Д\#} = SX.\text{Д\#}))$$

Равносильное этому выражение выглядит так:

$SX.\text{Город_П} \text{ WHERE NOT EXISTS PX (NOT EXISTS SPX}$

$$(\text{SPX.П\#} = SX.\text{П\#} \text{ AND} \\ \text{SPX.Д\#} = SX.\text{Д\#}))$$

Описанное исчисление не обладает вычислительной полнотой, так как не позволяет выполнять вычисления, связанные с обработкой данных в базах. Добавление вычислительных функций в это исчисление можно реализовать путем расширения определения операндов сравнения и элементов целевого списка таким образом, чтобы они допускали использование скалярных выражений с литералами, ссылками на атрибуты и итоговыми функциями. В качестве итоговых могут выступать следующие функции: COUNT (количество), SUM (сумма), AVG (среднее), MAX (максимальное), MIN (минимальное). Для целевых элементов целесообразно использовать спецификацию вида «AS <имя атрибута>», позволяющую явно задать имя результирующему атрибуту, если нет очевидного наследуемого имени.

Пример 4. Запись запроса.

Пусть требуется получить информацию о каждой поставке с полными данными о деталях и общем весе поставки. Запрос на дополненном функциями языке реляционного исчисления кортежей может выглядеть следующим образом:

$$(\text{SPX.П\#, SPX.Количество, PX, PX.Вес * SPX.Количество AS Общий_Вес}) \\ \text{WHERE PX.Д\#} = SPX.Д\#$$

Вариант реляционного исчисления, основанного на доменах (*исчисление доменов*), предложен Лакроиксом и Пиротте (Lacroix and Pirotte), которые также разработали на его основе соответствующий язык ILL. Другими языками, основанными на исчислении доменов, являются: FQL, DEDUCE, а также QBE с некоторыми оговорками.

По утверждению Дейта, язык QBE включает элементы исчисления кортежей и исчисления доменов, но более близок ко второму. Он не является реляционно полным, так как не поддерживает операцию отрицания квантора существования (NOT EXISTS). Несмотря на этот недостаток, язык QBE получил широкое распространение в современных СУБД. Тем более, что реализации этого языка, как правило, шире исходного языка.

Исчисление доменов имеет много сходства с исчислением кортежей. В отличие от исчисления кортежей, в *исчислении доменов* основой любого выражения запроса выступают переменные доменов. *Переменная домена* – это скалярная переменная, значения которой охватывают элементы некоторого домена.

Большая часть различий рассматриваемых исчислений заключается в том, что исчисление доменов поддерживает дополнительную форму условия, называемую *условием принадлежности*. В общем виде условие принадлежности записывается в виде:

$$R(A_1; \vartheta_1, A_2; \vartheta_2, \dots),$$

где A_i – атрибут отношения R , а ϑ_i – переменная домена или литерал. Проверяющее условие истинно, если и только если существует кортеж в отношении R , имеющий атрибуты A , равные заданным в выражении соответствующим значениям ϑ .

Например, выражение $SP(\Pi\# : 'S1', D\# : 'P1')$ истинно, если в отношении SP существует хотя бы один кортеж со значением 'S1' атрибута $\Pi\#$ и значением 'P1' атрибута $D\#$. Аналогично, выражение $SP(\Pi\# : SX, D\# : PX)$ истинно, если в отношении SP существует кортеж, в котором значение атрибута $\Pi\#$ эквивалентно текущему значению переменной домена SX , а значение атрибута $D\#$ эквивалентно текущему значению переменной домена PX .

В следующих примерах будем подразумевать существование (объявленное каким-либо образом, подобно оператору *RANGE* исчисления кортежей) следующих переменных доменов: SX (домен $\Pi\#$), PX (домен $D\#$), $NAMEX$ (домен Имя).

Пример 5. Выражения исчисления доменов.

```
(SX) WHERE S(\Pi\# : SX)
(SX) WHERE S(\Pi\# : SX, Город_\Pi : 'Москва')
NAMEX WHERE EXISTS SX ( S(\Pi\# : SX, Имя : NAMEX)
    AND FORALL PX ( IF P(D\# : PX)
        THEN SP(\Pi\# : SX, D\# : PX) ) )
```

Первое выражение означает множество всех номеров поставщиков отношения S , второе – множество номеров поставщиков из Лондона. Третье выражение соответствует запросу на получение имен поставщиков, производящих все детали.

3.8. Язык запросов по образцу QBE

Хранимые в базе данные можно обрабатывать *вручную*, последовательно просматривая и редактируя данные в таблицах с помощью имеющихся в СУБД соответствующих средств. Для повышения эффективности применя-

ют запросы, позволяющие производить *множественную* обработку данных, то есть одновременно вводить, редактировать и удалять множество записей, а также выбирать данные из таблиц.

Запрос представляет собой специальным образом описанное требование, определяющее состав производимых над БД операций по выборке, удалению или модификации хранимых данных.

Для подготовки запросов с помощью различных СУБД чаще всего используются два основных языка описания запросов:

- язык QBE (Query By Example) — язык запросов по образцу;
- SQL (Structured Query Language) — структурированный язык запросов.

По возможностям манипулирования данными при описании запросов указанные языки практически эквивалентны. Главное отличие между ними, по-видимому, заключается в способе формирования запросов: язык QBE предполагает ручное или визуальное формирование запроса, в то время как использование SQL означает программирование запроса.

Характеристика языка QBE

Теоретической основой языка QBE является реляционное исчисление с переменными-доменами. Язык QBE позволяет задавать сложные запросы к БД путем заполнения предлагаемой СУБД запросной формы. Такой способ задания запросов обеспечивает высокую наглядность и не требует указания алгоритма выполнения операции — достаточно описать образец ожидаемого результата. В каждой из современных реляционных СУБД имеется свой вариант языка QBE.

На языке QBE можно задавать запросы *однотабличные* и *многотабличные* (выбирающие или обрабатывающие данные из нескольких связанных таблиц).

С помощью запросов на языке QBE можно выполнять следующие основные операции:

- выборку данных;
- вычисление над данными;
- вставку новых записей;
- удаление записей;
- модификацию (изменение) данных.

Результатом выполнения запроса является новая таблица, называемая *ответной* (первые две операции), или обновленная исходная таблица (остальные операции).

Выборка, вставка, удаление и модификация могут производиться безусловно или в соответствии с условиями, задаваемыми с помощью логических выражений. Вычисления над данными задаются с помощью арифметических выражений и порождают в ответных таблицах новые поля, называемые *вычисляемыми*.

Запросная форма имеет вид таблицы, имя и названия полей которой совпадают с именем и названиями полей соответствующей исходной таблицы. Чтобы узнать имена доступных таблиц БД, в языке QBE предусмотрен запрос на выборку имен таблиц. Названия полей исходной таблицы могут вводиться в шаблон вручную или автоматически. Во втором случае используется запрос на выборку заголовков столбцов.

В современных СУБД, например в Access и Visual FoxPro, многие действия по подготовке запросов с помощью языка QBE выполняются визуально с помощью мыши. В частности, визуальное связывание таблиц при подготовке запроса выполняется не элементами примеров, а просто «протаскиванием» мышью поля одной таблицы к полю другой.

Первоначальный вариант QBE

В большинстве современных СУБД имеется свой вариант QBE, незначительно отличающийся от первого описания QBE, предложенного Злуфом М.М. в 1975–1977 гг. Рассмотрим основные возможности QBE, опираясь на первоначальный его вариант.

Для иллюстрации средств и возможностей языка QBE используем таблицы БД, которая относится к торговой сфере и используется в фирме, продающей товары нескольких видов. База данных включает четыре следующие таблицы с полями:

- EMP (служащие): ФИО – фамилия и инициалы служащего, ЗАРПЛАТА – размер должностного оклада, РУКОВОДИТЕЛЬ – фамилия и инициалы руководителя, ОТДЕЛ – название отдела, в котором работает служащий;
- SALES (продажи): ОТДЕЛ – название отдела, ТОВАР – название товара;
- SUPPLY (поставщики): ТОВАР – название поставляемого товара, ПОСТАВЩИК – название организации, поставляющей товар;
- TYPE (типы товаров): ТОВАР – название товара, ЦВЕТ – его цвет, СТОИМОСТЬ – стоимость товара.

В таблицах приведены неполные и упрощенные сведения. Так, в таблице TYPE указаны не все виды товаров, приведенные в таблице SALES.

Для описания переменных в условиях отбора записей, а также для связывания шаблонов в запросах используются элементы примера.

Элемент примера играет роль идентификатора переменной (как в языке программирования) и задается с помощью символьно-цифровой последовательности. Элементы примера в шаблонах выделим подчеркиванием. Вид (длина и состав) элемента примера роли не играют: главное чтобы при использовании в нескольких местах шаблона он был одинаков. Таким образом, как элементы примера, в частности, можно использовать идентификаторы example, x или y.

Для указания системе необходимости включения в ответную таблицу того или иного поля используется «P.», что означает «напечатать».

EMP	ФИО	ЗАРПЛАТА	РУКОВОДИТЕЛЬ	ОТДЕЛ
	Киселев В.М.	1800	Белкин Б.Н.	хозтовары
	Гурский С.И.	1600	Томилов А.Н.	игрушки
	Андреева Е.А.	2000	Петров А.С.	косметика
	Левин П.Г.	2200	Петров А.С.	канцтовары
	Носов А.П.	1600	Томилов А.Н.	игрушки
	Гофман В.Э.	2600	Андреева Е.А.	косметика
	Сорокина Т.В.	1700	Андреева Е.А.	косметика
	Белкин Б.Н.	1800	Петров А.С.	хозтовары
	Семин С.В.	2200	Левин П.Г.	канцтовары
	Григорьев А.Н.	1900	Томилов А.Н.	игрушки
	Томилов А.Н.	2000	Петров А.С.	игрушки

SALES	ОТДЕЛ	ТОВАР	SUPPLY	ТОВАР	ПОСТАВЩИК
	канцтовары	бумага		ручка	Pencraft
	хозтовары	мыло		бумага	Pencraft
	канцтовары	карандаш		мыло	Procter & Gamble
	косметика	помада		карандаш	Flic
	игрушки	самолет		чернила	Pencraft
	игрушки	машина		духи	Beautex
	игрушки	кукла		чернила	Flic
	косметика	духи		посуда	Cremco
	канцтовары	чернила		помада	Beautex
	хозтовары	посуда		самолет	Signal
	канцтовары	ручка		машина	Signal
				кукла	Signal
				посуда	Flic
				ручка	Beautex
				карандаш	Pencraft

TYPE	ТОВАР	ЦВЕТ	СТОИМОСТЬ
	посуда	белый	30
	помада	красный	17
	духи		42
	ручка	зеленый	6
	карандаш	синий	2
	чернила	зеленый	4
	чернила	синий	3
	карандаш	красный	2
	карандаш	синий	2

Пример 1. Запрос на выборку.

С учетом изложенного запрос на выборку всех зеленых товаров можно записать в следующем виде:

TYPE	ТОВАР	ЦВЕТ	СТОИМОСТЬ
	P. XX	зеленый	

Словесно запрос можно сформулировать следующим образом: «Вывести все товары **XX**, цвет которых зеленый». В приведенном шаблоне элемент **XX** не обязательен и его можно опустить. Элементы примера указываются обязательно при записи логических условий, а также при связывании таблиц в запросах.

Пустые колонки из шаблона можно удалять.

Пример 2. Удаление колонок.

У нас неиспользуемым столбцом является СТОИМОСТЬ. Исходя из этого, для приведенного шаблона можно записать следующий эквивалентный шаблон:

TYPE	ТОВАР	ЦВЕТ
	P.	зеленый

После заполнения шаблона для получения результата нажимается соответствующая клавиша, например <Enter>, и начинается выполнение запроса. Результатом выполнения приведенного запроса является следующая таблица:

TYPE	ТОВАР
	ручка
	чернила

Рассмотрим основные возможности перечисленных выше типов операций, используемых в запросах.

Выборка данных

Простая выборка. Примером простой выборки является запрос: «Вывести все возможные цвета товаров из таблицы TYPE».

Пример 3. Простая выборка.

Заполненный шаблон в этом случае будет выглядеть так:

TYPE	ТОВАР	ЦВЕТ	СТОИМОСТЬ
		P.	

Ответная таблица имеет единственный столбец ЦВЕТ, содержащий значения: белый, красный, пусто (значение не задано), зеленый и синий. Дублируемые значения при этом пропадают.

Если требуется вывести данные из нескольких полей исходной таблицы, в каждом из соответствующих столбцов шаблона записывается «P.». Занесение «P.» во все столбцы шаблона можно заменить записью «P.» в первом столбце шаблона под именем таблицы.

Простая выборка с упорядочиванием. Для упорядочения выводимых значений по возрастанию и по убыванию используются конструкции «AO.» и «DO.» соответственно. Если требуется выполнить упорядочивание по нескольким столбцам, применяют конструкции вида: «AO(1).», «AO(2).».

Выборка с квалификаторами (условиями). Выбор записей из исходной таблицы в общем случае может быть основан на: точном совпадении, частичном совпадении, сравнении.

1. *Точное совпадение* задается вводом констант в соответствующих полях шаблона, как в случае запроса по товарам зеленого цвета.

2. *Частичное совпадение* задается с помощью элементов примера. В частности, для формулировки запроса о выводе всех видов товаров, названия которых начинаются с буквы «и», можно воспользоваться конструкцией «P.i_ke», записанной в поле ТОВАР таблицы TYPE. Здесь: «P.» задает вывод, «и» — константа, а «_ke» — элемент примера, играющий роль переменной.

Используя элементы примера, можно задавать различные варианты частичного совпадения со значениями данных из таблиц: в начале «i_ke», в конце «_xa», в середине «x1ox2» и в произвольном месте.

Поскольку элементу примера сопоставим любой символ, в том числе и пустой (отсутствие символа), то условию частичного совпадения «x1ox2» удовлетворяют слова, не только имеющие символ «o» в середине, но начинаящиеся и заканчивающиеся на «o».

Пример 4. Частичное совпадение.

Шаблон запроса с выбором товаров синего цвета, в середине названий которых имеется буква «р», выглядит так:

TYPE	ТОВАР	ЦВЕТ	СТОИМОСТЬ
	P.хру	синий	

Результат в этом случае будет следующий:

TYPE	ТОВАР
	карандаш
	чернила

3. Условие сравнения записывается с помощью операций сравнения: равно (=), больше (>), меньше (<), больше или равно (>=), меньше или равно (<=), не равно (\neq или просто \neg), не больше ($\neg>$), не меньше ($\neg<$).

Пример 5. Условия сравнения.

Запрос имен сотрудников, работающих в отделе игрушек и получающих зарплату больше 1800, выглядит так:

EMP	ФИО	ЗАРПЛАТА	РУКОВОДИТЕЛЬ	ОТДЕЛ
	P.	>1800		игрушки

Результатом запроса является таблица вида:

EMP	ФИО
	Григорьев А.Н.
	Томилов А.Н.

В операциях сравнения можно использовать и элементы примера.

Пример 6. Сравнения с элементами примера.

Ниже приведен вид шаблона запроса выборки имен и зарплат сотрудников, получающих больше, чем Левин П.Г. По-другому запрос можно сформулировать так: «Пусть Левин П.Г. получает зарплату в количестве s . Найти всех сотрудников, получающих больше, чем s , и вывести их зарплаты». Порядок строк в шаблоне несущественен.

EMP	ФИО	ЗАРПЛАТА
	P.	$P > s$
	Левин П.Г.	s

Сотрудников, получающих больше Левина П.Г., в таблице EMP не оказалось (вероятно, таковым является Петров А.С., но из таблицы EMP нам этого не узнать). В подобных случаях результирующая таблица оказывается пустой:

EMP	ФИО	ЗАРПЛАТА

Условия в запросе могут задаваться по одному или по нескольким столбцам. При этом происходит объединение отдельных условий по схеме логического И (AND).

Пример 7. Объединение условий.

Для формулировки запроса выборки имен и зарплат служащих, получающих больше Левина П.Г. и работающих в отделе игрушек, достаточно в предыдущую запросную форму в первую строку столбца ОТДЕЛ вставить слово «игрушки».

Запрос вида: «Найти имена и зарплаты служащих, получающих больше, чем Белкин Б.Н., и работающих в отделе, продающем ручки», выглядит следующим образом:

EMP	ФИО	ЗАРПЛАТА	РУКОВОДИТЕЛЬ	ОТДЕЛ
	P.	P.> S		department
	Белкин Б.Н.	S		

SALES	ОТДЕЛ	ТОВАР
	department	ручка

Результатом выполнения этого запроса будет таблица вида:

EMP	ФИО	ЗАРПЛАТА
	Левин П.Г.	2200
	Семин С.В.	2200

Здесь элемент примера department связывает две исходные таблицы по полю ОТДЕЛ, а элемент примера s используется для связи условий выбора в рамках одной исходной таблицы EMP.

Пример 8. Запрос, в шаблоне которого имеются две связи.

Пусть необходимо найти всех служащих, получающих больше своих руководителей. Этот запрос с помощью элементов примеров можно сформулировать так: «Вывести всех служащих, чьи руководители являются head и кто

получает больше, чем s , где s — зарплата `head`. Шаблон соответствующего запроса имеет следующий вид:

EMP	ФИО	ЗАРПЛАТА	РУКОВОДИТЕЛЬ	ОТДЕЛ
	P.	> s	<code>head</code>	
	<code>head</code>	s		

Здесь элемент `head` используется для связи руководителя в первой строке шаблона и имени из второй строки, а элемент s применяется для сравнения зарплат.

Для нашего примера результирующая таблица окажется такой:

EMP	ФИО
	Гофман В.Э.

В качестве условия выбора записей из таблиц можно использовать операцию отрицания.

Пример 9. Отбор с операцией отрицания.

Пусть необходимо вывести все отделы, продающие товары, не поставляемые компанией Pencraft. Этот запрос можно перефразировать: «Вывести названия отделов, продающих товары t , такие, что компания Pencraft не поставляет товары t ».

SALES	ОТДЕЛ	ТОВАР
	P.	t

SUPPLY	ТОВАР	ПОСТАВЩИК
	$\neg t$	Pencraft

Ответная таблица для сформулированного запроса имеет вид:

SALES	ОТДЕЛ
	хозтовары
	косметика
	игрушки

В случаях, когда условия отбора записей для выборки представляют большие выражения, которые неудобно или трудно задать в шаблоне, можно использовать **блок условий**. Он по виду напоминает пустой шаблон с одним полем и именем CONDITIONS. Блок условий предназначен для записи логических выражений.

Записанные в одном шаблоне логические выражения, в общем случае, могут включать в себя операции логического умножения (операция AND) и логического сложения (операция OR).

Пример 10. Использование блока условий.

Шаблон запроса вывода фамилий сотрудников, чья зарплата составляет от 2000 до 2500, но не равна 2300, будет выглядеть так:

EMP	ФИО	ЗАРПЛАТА
	P. Jon	>2000
	Jon	<2500
	Jon	=2300

Используя блок условий с явным заданием операции AND (символ &), этот же запрос можно сформулировать проще:

EMP	ФИО	ЗАРПЛАТА
	P.	s

CONDITIONS
<u>s</u> = (>2000 & <2500 & ≠2300)

Шаблон запроса вывода фамилий сотрудников, чья зарплата составляет 2000, 2300 или 2600, можно сформировать так:

EMP	ФИО	ЗАРПЛАТА
	P. Jon	2000
	P. Mak	2300
	P. Nik	2600

В каждой из строк шаблона используются различные элементы примера, и поэтому условия действуют независимо. С помощью блока условий, в котором операция OR (символ |) задана явно, этот же запрос будет выглядеть более наглядно:

EMP	ФИО	ЗАРПЛАТА
	P.	s

CONDITIONS
<u>s</u> = (2000 2300 2600)

При записи логических выражений на языке QBE могут применяться **встроенные функции**, такие как: CNT. (счетчик или количество), SUM. (сумма), AVG. (среднее), MIN. (минимум), MAX. (максимум), UN. (уникальный)

и ALL. (все значения, в том числе повторяющиеся). Первые пять из них являются статистическими, а последние две определяют характер выборки: включать или не включать в выборку повторяющиеся значения.

Функцию UN. можно присоединять к функциям CNT., SUM. и AVG.. Так, запись CNT.UN. означает количество только различающихся значений. В противоположность этому, запись CNT.ALL. будет означать количество всех значений. Очевидно, функции MAX.UN. и MAX.ALL. дадут одинаковый результат.

Пример 11. Использование функций.

Пусть требуется вывести названия отделов, в которых работает более двух сотрудников. Этот запрос можно разделить на три операции: сгруппировать сотрудников по отделам, подсчитать число сотрудников в каждом из отделов и отобрать отделы, в которых работает более двух сотрудников.

EMP	ФИО	ОТДЕЛ
	ALL.Employee	P. Toy

CONDITIONS
CNT.ALL.Employee > 2

Конструкция **Toy** означает операцию **группирования** (group-by), функция ALL.Employee формирует множество (точнее мультимножество, так как допускает дублирования) всех имен по каждому из отделов, а запись CNT.ALL.Employee > 2 обеспечивает проверку логического условия. Здесь встретилась новая операция формирования группы, которая записывается с помощью выделенного жирным шрифтом элемента примера (в оригиналe – двойное подчеркивание элемента примера).

Результирующая таблица имеет вид:

EMP	ОТДЕЛ
	игрушки
	косметика

Другой пример. Вывести названия отделов, в которых продаются только товары зеленого цвета.

SALES	ОТДЕЛ	ТОВАР
	P. Toy	ALL.Ink

TYPE	ТОВАР	ЦВЕТ
	ALL.Ink	зеленый

Ответная таблица для данного запроса будет пустой.

Вычисления в запросах

С помощью запросов можно выбирать данные из таблиц и производить вычисления. Вид вычисления задается с помощью выражения в шаблоне. В выражениях, помимо обычных арифметических операций (+, -, *, /) и скобок, могут использоваться встроенные функции: AVG., CNT., MAX., MIN. и SUM..

Примеры 12. Шаблоны с вычислениями.

Пусть имеется таблица EMP1 с полями имени (ФИО), размером зарплаты (ЗАРПЛАТА) и размером премиальных (ПРЕМИЯ). Необходимо по каждому из сотрудников вывести имя и общую сумму зарплаты и премиальных. Для этого сформируем шаблон новой таблицы OUTPUT (заполнив строку имени таблицы и имен ее полей) и укажем в ней вид вычислений. Связем этот шаблон с шаблоном запроса к таблице EMP1.

OUTPUT	ФИО	СУММА
P. Employee		P. (s1+s2)
EMP1	ФИО	ЗАРПЛАТА
	Employee	s1
		s2

Поскольку операция суммирования выполняется по каждой строке исходной таблицы, такой тип вычислений можно назвать *горизонтальным*. Встроенные функции оперируют группами записей, поэтому можно считать, что они выполняют *вертикальные вычисления*.

Для подсчета общего числа сотрудников следует составить такой запрос:

EMP1	ФИО	ОТДЕЛ
	P.CNT.ALL. Em	

Элемент примера Em здесь можно опустить.

Если требуется подсчитать число сотрудников в отделе игрушек, следует подготовить шаблон запроса вида:

EMP1	ФИО	ОТДЕЛ
	P.CNT.ALL. Em	игрушки

Чтобы подсчитать число сотрудников в каждом из отделов (с выводом наимений отделов), подготовим шаблон запроса вида:

EMP1	ФИО	ОТДЕЛ
	P.CNT.ALL. Em	P. Dep

В этом случае получим следующий результат:

EMP1	ФИО	ОТДЕЛ
	2	хозтовары
	3	игрушки
	3	косметика
	2	канцтовары

В шаблоне запроса мы применили операцию группирования (см. Пример 11).

Операции вставки, удаления и модификации

В отличие от рассмотренных операций, операции вставки, удаления и модификации приводят к *изменению* исходной таблицы. Вид операции (вставка – I., удаление – D., модификация – U.) записывается в шаблоне под именем таблицы, а константы и условные выражения указываются по тем же правилам, что и в операциях выборки.

Примеры 13. Операции вставки, удаления, модификации.

Для вставки в таблицу EMP нового сотрудника отдела игрушек с фамилией Деревянко Н.В., зарплатой 2000 и руководителем Белкиным Б.Н. нужно сформировать шаблон вида:

EMP	ФИО	ЗАРПЛАТА	РУКОВОДИТЕЛЬ	ОТДЕЛ
I.	Деревянко Н.В.	2000	Белкин Б.Н.	игрушки

Пусть необходимо удалить всю информацию о сотрудниках отдела игрушек. Шаблон соответствующего запроса будет выглядеть так:

EMP	ФИО	ЗАРПЛАТА	РУКОВОДИТЕЛЬ	ОТДЕЛ
D.				игрушки

Для модификации некоторого значения отдельного поля, например размера зарплаты сотрудника Белкина Б.Н., достаточно сформировать следующий шаблон:

EMP	ФИО	ЗАРПЛАТА	РУКОВОДИТЕЛЬ	ОТДЕЛ
U.	Белкин Б.Н.	2100		

Пустое поле означает, что оно не подлежит изменению. Если требуется изменить некоторое значение на «пустое», используется ключевое слово NULL.

Чтобы повысить зарплату сотрудникам отдела игрушек на 10%, можно сформировать шаблон запроса на модификацию следующего вида:

EMP	ФИО	ЗАРПЛАТА	РУКОВОДИТЕЛЬ	ОТДЕЛ
U.		1.1*s1		
		s1		игрушки

Реализация этого запроса происходит в два этапа: сначала выбираются все записи со значением «игрушки» в поле ОТДЕЛ, а затем происходит изменение поля ЗАРПЛАТА отобранных записей на новое значение.

Характеристика языков QBE современных СУБД

Основные отличия языков QBE современных СУБД от языка, предложенного Злуфром М.М., как правило, сводятся к незначительным изменениям в интерпретации отдельных реляционных операций, введению дополнительных операций и изменению формы представления языка.

Например, в системе Paradox for Windows вместо операции печати Р. применен метод отметки выбираемых в запросной форме (шаблоне) полей. Для этого в начале каждого из полей запросной формы располагаются флагки для выбора поля. Отмечая поля, пользователь может указать последовательность сортировки в ответной таблице. Для связывания нескольких запросных форм в один многотабличный запрос, а также в логических выражениях условий отбора записей применяются *элементы примера*.

Наглядными являются запросные формы в Microsoft Access. Диалоговое окно (рис. 3.9) при подготовке запросных форм состоит из двух частей: в верхней части располагается модель взаимосвязи исходных таблиц, а в нижней — остальная информация о запросе по каждому из полей (необходимость вывода значений, вид сортировки, условие отбора и т. д.).

Подготовка шаблона запроса выполняется пользователем с помощью мыши. Так, связывание таблиц в запросе производится не элементами примеров, а «буксировкой» поля одной таблицы к полю другой таблицы. Если таблицы имели связь между собой, то система автоматически связывает все находящиеся в запросной форме таблицы. При этом каждая из связей помечается в соответствии с ее типом (на рис. 3.9 между таблицами Поставщики и Товары по полю КодПоставщика образована связь вида 1:М).

Анализ современных СУБД позволяет предположить следующие направления развития языка QBE:

- Повышение наглядности и удобства.
- Появление средств, соответствующих новым возможностям СУБД, например, формулировка неточных или нечетких запросов, манипулирование большими объемами данных.
- Использование новых типов данных (графических, аудио-, видео- и др.).

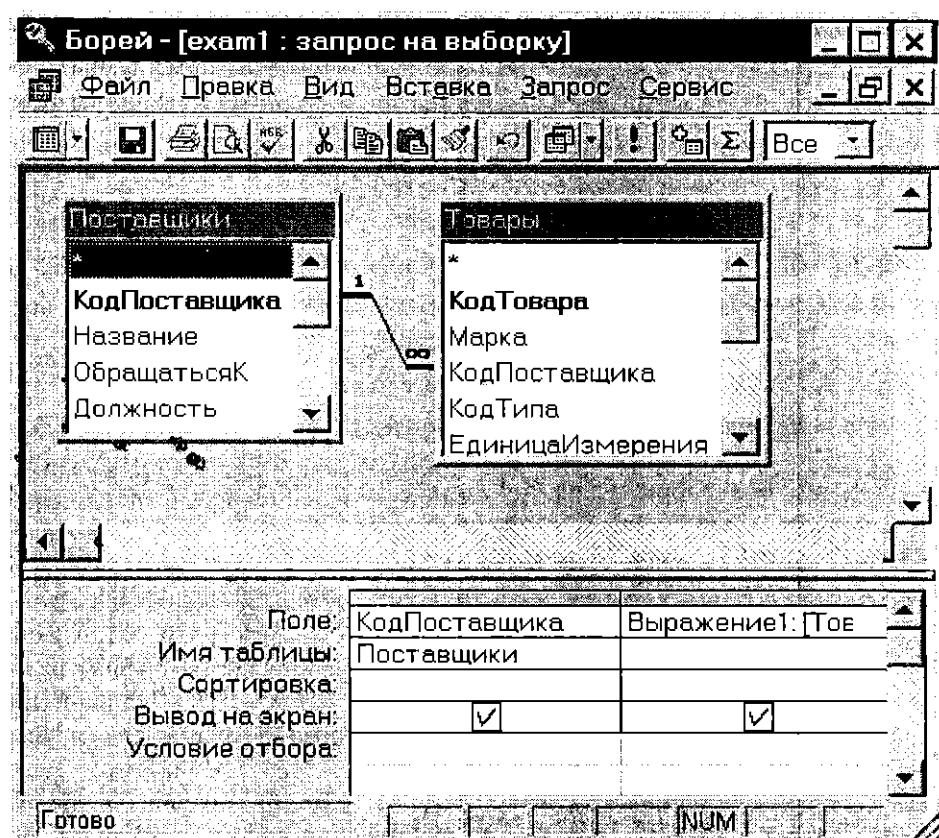


Рис. 3.9. Вид запросной формы в Microsoft Access

- Применение в ближайшем будущем ограниченного естественного языка формулировки запросов.
- В более отдаленной перспективе использование речевого ввода запросов.

Принципиальные возможности для перехода к естественному языку общения и средствам речевого ввода есть уже сегодня. Это можно сделать, например, в виде надстройки над существующими СУБД при использовании словарей соответствия терминов ограниченного естественного языка и названий таблиц БД, полей таблиц, операций над данными и другими элементами QBE.

3.9. Структурированный язык запросов SQL

Структурированный язык запросов SQL основан на реляционном исчислении с *переменными кортежами*. Язык имеет несколько стандартов, наиболее распространенными из которых являются SQL-89 и SQL-92 (подраздел 9.3).

Общая характеристика языка

Язык SQL предназначен для выполнения операций над таблицами (создание, удаление, изменение структуры) и над данными таблиц (выборка, изменение, добавление и удаление), а также некоторых сопутствующих операций. SQL является *непроцедурным языком* и не содержит операторов управления, организации подпрограмм, ввода-вывода и т. п. В связи с этим SQL автономно не используется, обычно он погружен в среду встроенного языка программирования СУБД (например, FoxPro СУБД Visual FoxPro, ObjectPAL СУБД Paradox, Visual Basic for Applications СУБД Access).

В современных СУБД с интерактивным интерфейсом можно создавать запросы, используя другие средства, например QBE. Однако применение SQL зачастую позволяет повысить эффективность обработки данных в базе. Например, при подготовке запроса в среде Access можно перейти из окна Конструктора запросов (формулировки запроса по образцу на языке QBE) в окно с эквивалентным оператором SQL. Подготовку нового запроса путем редактирования уже имеющегося в ряде случаев проще выполнить путем изменения оператора SQL. В различных СУБД состав операторов SQL может несколько отличаться.

Язык SQL не обладает функциями полноценного языка разработки, а ориентирован на доступ к данным, поэтому его включают в состав средств разработки программ. В этом случае его называют *встроенным SQL*. Стандарт языка SQL поддерживают современные реализации следующих языков программирования: PL/1, Ada, C, COBOL, Fortran, MUMPS и Pascal.

В специализированных системах разработки приложений типа клиент-сервер среда программирования, кроме того, обычно дополнена коммуникационными средствами (установление и разъединение соединений с серверами БД, обнаружение и обработка возникающих в сети ошибок и т. д.), средствами разработки пользовательских интерфейсов, средствами проектирования и отладки.

Различают два основных метода использования встроенного SQL: статический и динамический.

При *статическом* использовании языка (*статический SQL*) в тексте программы имеются вызовы функций языка SQL, которые жестко включаются в выполняемый модуль после компиляции. Изменения в вызываемых функциях могут быть на уровне отдельных параметров вызовов с помощью переменных языка программирования.

При *динамическом* использовании языка (*динамический SQL*) предполагается динамическое построение вызовов SQL-функций и интерпретация этих вызовов, например, обращение к данным удаленной базы, в ходе выполнения программы. Динамический метод обычно применяется в случаях, когда в приложении заранее неизвестен вид SQL-вызыва и он строится в диалоге с пользователем.

Основным назначением языка SQL (как и других языков для работы с базами данных) является подготовка и выполнение запросов. В результате выборки данных из одной или нескольких таблиц может быть получено множество записей, называемое *представлением*.

Представление по существу является таблицей, формируемой в результате выполнения запроса. Можно сказать, что оно является разновидностью хранимого запроса. По одним и тем же таблицам можно построить несколько представлений. Само представление описывается путем указания идентификатора представления и запроса, который должен быть выполнен для его получения.

Для удобства работы с представлениями в языке SQL введено понятие курсора. **Курсор** представляет собой своеобразный указатель, используемый для перемещения по наборам записей при их обработке.

Описание и использование курсора в языке SQL выполняется следующим образом. В описательной части программы выполняют связывание переменной типа курсор (CURSOR) с оператором SQL (обычно с оператором SELECT). В выполняемой части программы производится открытие курсора (OPEN <имя курсора>), перемещение курсора по записям (FETCH <имя курсора>...), сопровождаемое соответствующей обработкой, и, наконец, закрытие курсора (CLOSE <имя курсора>).

Основные операторы языка

Опишем минимальное подмножество языка SQL, опираясь на его реализацию в стандартном интерфейсе ODBC (Open Database Connectivity — совместимость открытых баз данных) фирмы Microsoft.

Операторы языка SQL можно условно разделить на два подъязыка: язык определения данных (Data Definition Language — DDL) и язык манипулирования данными (Data Manipulation Language — DML). Основные операторы языка SQL представлены в табл. 3.3.

Рассмотрим формат и основные возможности важнейших операторов, за исключением специфических операторов, отмеченных в таблице символом «*». Несущественные операнды и элементы синтаксиса (например, принятые во многих системах программирования правило ставить «;» в конце оператора) будем опускать.

1. Оператор **создания таблицы** имеет формат вида:

```
CREATE TABLE <имя таблицы>
    (<имя столбца> <тип данных> [NOT NULL]
     [,<имя столбца> <тип данных> [NOT NULL]] ... )
```

Обязательными operandами оператора являются имя создаваемой таблицы и имя хотя бы одного столбца (поля) с указанием типа данных, хранимых в этом столбце.

При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Конструкция NOT NULL (не пустое) служит именно таким целям и для столбца таблицы означает, что в этом столбце должно быть определено значение.

Таблица 3.3

Операторы языка SQL

Вид	Название	Назначение
DDL	CREATE TABLE	создание таблицы
	DROP TABLE	удаление таблицы
	ALTER TABLE	изменение структуры таблицы
	CREATE INDEX	создание индекса
	DROP INDEX	удаление индекса
	CREATE VIEW	создание представления
	DROP VIEW	удаление представления
	GRANT*	назначение привилегий
	REVOKE*	удаление привилегий
DML	SELECT	выборка записей
	UPDATE	изменение записей
	INSERT	вставка новых записей
	DELETE	удаление записей

В общем случае в разных СУБД могут использоваться различные типы данных (см. подраздел 2.7). В интерфейсе ODBC поддерживаются свои стандартные типы данных, например, символьные (SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR) и др. При работе с БД некоторой СУБД посредством интерфейса ODBC выполняется автоматическое преобразование стандартных типов данных, поддерживаемых интерфейсом, в типы данных источников и обратно. При необходимости обмен данными между программой и источником данных может вестись без преобразования — во внутреннем формате данных источника.

Пример 1. Создание таблицы.

Пусть требуется создать таблицу goods описания товаров, имеющую поля: type — вид товара, comp_id — идентификатор компании-производителя, name — название товара и price — цена товара. Оператор определения таблицы может иметь следующий вид:

```
CREATE TABLE goods (type SQL_CHAR(8) NOT NULL,
                    comp_id SQL_CHAR(10) NOT NULL, name SQL_VARCHAR(20),
                    price SQL_DECIMAL(8,2)).
```

2. Оператор *изменения структуры таблицы* имеет формат вида:

```
ALTER TABLE <имя таблицы>
  ( {ADD, MODIFY, DROP} <имя столбца> [<тип данных>]
    [NOT NULL]
  , {ADD, MODIFY, DROP} <имя столбца> [<тип данных>]
    [NOT NULL] ... )
```

Изменение структуры таблицы может состоять в добавлении (ADD), изменении (MODIFY) или удалении (DROP) одного или нескольких столбцов таблицы. Правила записи оператора ALTER TABLE такие же, как и оператора CREATE TABLE. При удалении столбца указывать <тип данных> не нужно.

Пример 2. Добавление поля таблицы.

Пусть в созданной ранее таблице goods необходимо добавить поле number, отводимое для хранения величины запаса товара. Для этого следует записать оператор вида:

```
ALTER TABLE goods (ADD number SQL_INTEGER).
```

3. Оператор **удаления таблицы** имеет формат вида:

```
DROP TABLE <имя таблицы>
```

Оператор позволяет удалить имеющуюся таблицу. Например, для удаления таблицы с именем items достаточно записать оператор вида:

```
DROP TABLE items.
```

4. Оператор **создания индекса** имеет формат вида:

```
CREATE [UNIQUE] INDEX <имя индекса>
    ON <имя таблицы>
        (<имя столбца> [ASC|DESC]
        [,<имя столбца> [ASC|DESC] ... ])
```

Оператор позволяет создать индекс для одного или нескольких столбцов заданной таблицы с целью ускорения выполнение запросных и поисковых операций с таблицей. Для одной таблицы можно создать несколько индексов.

Задав необязательную опцию UNIQUE, можно обеспечить уникальность значений во всех указанных в операторе столбцах. По существу, создание индекса с указанием признака UNIQUE означает определение ключа в созданной ранее таблице.

При создании индекса можно задать порядок автоматической сортировки значений в столбцах — в порядке возрастания ASC (по умолчанию), или в порядке убывания DESC. Для разных столбцов можно задавать различный порядок сортировки.

Пример 3. Создание индекса.

Пусть для таблицы EMP, имеющей поля: NAME (имя), SAL (зарплата), MGR (руководитель) и DEPT (отдел), нужно создать индекс main_indx для сортировки имен в алфавитном порядке и убыванию размеров зарплаты. Оператор создания индекса может иметь вид:

```
CREATE INDEX main_indx
    ON emp (name, sal DESC).
```

5. Оператор **удаления индекса** имеет формат вида:

```
DROP INDEX <имя индекса>
```

Этот оператор позволяет удалять созданный ранее индекс с соответствующим именем. Так, например, для уничтожения индекса main_indx к таблице emp достаточно записать оператор DROP INDEX main_indx.

6. Оператор **создания представления** имеет формат вида:

```
CREATE VIEW <имя представления>
[(<имя столбца> [,<имя столбца> ] ... )]
AS <оператор SELECT>
```

Данный оператор позволяет создать представление. Если имена столбцов в представлении не указываются, то будут использоваться имена столбцов из запроса, описываемого соответствующим оператором SELECT.

Пример 4. Создание представления.

Пусть имеется таблица companies описания производителей товаров с полями: comp_id (идентификатор компании), comp_name (название организации), comp_address (адрес) и phone (телефон), а также таблица goods производимых товаров с полями: type (вид товара), comp_id (идентификатор компании), name (название товара) и price (цена товара). Таблицы связаны между собой по полю comp_id. Требуется создать представление герг с краткой информацией о товарах и их производителях: вид товара, название производителя и цена товара. Оператор определения представления может иметь следующий вид:

```
CREATE VIEW
    герг
AS
    SELECT
        goods.type, companies.comp_name, goods.price
    FROM
        goods, companies
    WHERE
        goods.comp_id = companies.comp_id
```

7. Оператор **удаления представления** имеет формат вида:

```
DROP VIEW <имя представления>
```

Оператор позволяет удалить созданное ранее представление. Заметим, что при удалении представления таблицы, участвующие в запросе, удалению не подлежат. Удаление представления герг производится оператором вида:

```
DROP VIEW герг.
```

8. Оператор **выборки записей** имеет формат вида:

```
SELECT [ALL | DISTINCT]
    <список данных>
    FROM <список таблиц>
```

[WHERE <условие выборки>]
 [GROUP BY <имя столбца> [<имя столбца>] ...]
 [HAVING <условие поиска>]
 [ORDER BY <спецификация> [<спецификация>] ...]

Это наиболее важный оператор из всех операторов SQL. Функциональные возможности его огромны. Рассмотрим основные из них.

Оператор SELECT позволяет производить выборку и вычисления над данными из одной или нескольких таблиц. Результатом выполнения оператора является ответная таблица, которая может иметь (ALL), или не иметь (DISTINCT) повторяющиеся строки. По умолчанию в ответную таблицу включаются все строки, в том числе и повторяющиеся. В отборе данных участвуют записи одной или нескольких таблиц, перечисленных в списке операнда FROM.

Список данных может содержать имена столбцов, участвующих в запросе, а также выражения над столбцами. В простейшем случае в выражениях можно записывать имена столбцов, знаки арифметических операций (+, -, *, /), константы и круглые скобки. Если в списке данных записано выражение, то наряду с выборкой данных выполняются вычисления, результаты которого попадают в новый (создаваемый) столбец ответной таблицы.

При использовании в списках данных имен столбцов нескольких таблиц для указания принадлежности столбца некоторой таблице применяют конструкцию вида: <имя таблицы>.<имя столбца>.

Операнд WHERE задает условия, которым должны удовлетворять записи в результирующей таблице. Выражение <условие выборки> является логическим. Его элементами могут быть имена столбцов, операции сравнения, арифметические операции, логические связки (И, ИЛИ, НЕТ), скобки, специальные функции LIKE, NULL, IN и т. д.

Операнд GROUP BY позволяет выделять в результирующем множестве записей группы. *Группой* являются записи с совпадающими значениями в столбцах, перечисленных за ключевыми словами GROUP BY. Выделение групп требуется для использования в логических выражениях operandов WHERE и HAVING, а также для выполнения операций (вычислений) над группами.

В логических и арифметических выражениях можно использовать следующие групповые операции (функции): AVG (среднее значение в группе), MAX (максимальное значение в группе), MIN (минимальное значение в группе), SUM (сумма значений в группе), COUNT (число значений в группе).

Операнд HAVING действует совместно с операндом GROUP BY и используется для дополнительной селекции записей во время определения групп. Правила записи <условия поиска> аналогичны правилам формирования <условия выборки> операнда WHERE.

Операнд ORDER BY задает порядок сортировки результирующего множества. Обычно каждая <спецификация> аналогична соответствующей конструкции оператора CREATE INDEX и представляет собой пару вида: <имя столбца> [ASC | DESC].

Замечание.

Оператор SELECT может иметь и другие более сложные синтаксические конструкции, которые мы подробно рассматривать не будем, а поясним их смысл.

Одной из таких конструкций, например, являются так называемые ***подзапросы***. Они позволяют формулировать ***вложенные запросы***, когда результаты одного оператора SELECT используются в логическом выражении условия выборки операнда WHERE другого оператора SELECT.

Вторым примером более сложной формы оператора SELECT является оператор, в котором отобранные записи в дальнейшем предполагается модифицировать (конструкция FOR UPDATE OF). СУБД после выполнения такого оператора обычно блокирует (защищает) отобранные записи от модификации их другими пользователями.

Еще один случай специфического использования оператора SELECT – выполнение объединений результирующих таблиц при выполнении нескольких операторов SELECT (операнд UNION).

Пример 5. Выбор записей.

Для таблицы EMP, имеющей поля: NAME (имя), SAL (зарплата), MGR (руководитель) и DEPT (отдел), требуется вывести имена сотрудников и размер их зарплаты, увеличенный на 100 единиц. Оператор выбора можно записать следующим образом:

```
SELECT name, sal+100  
      FROM emp.
```

Пример 6. Выбор с условием.

Вывести названия таких отделов таблицы EMP, в которых в данный момент отсутствуют руководители. Оператор SELECT для этого запроса можно записать так:

```
SELECT dept  
      FROM emp  
     WHERE mgr is NULL.
```

Пример 7. Выбор с группированием.

Пусть требуется найти минимальную и максимальную зарплаты для каждого из отделов (по таблице EMP). Оператор SELECT для этого запроса имеет вид:

```
SELECT dept, MIN(sal), MAX(sal)
      FROM emp
    GROUP BY dept.
```

9. Оператор ***изменения записей*** имеет формат вида:

```
UPDATE <имя таблицы>
    SET <имя столбца> = {<выражение>, NULL}
    [, SET <имя столбца> = {<выражение>, NULL} ... ]
    [WHERE <условие>]
```

Выполнение оператора UPDATE состоит в изменении значений в определенных операндом SET столбцах таблицы для тех записей, которые удовлетворяют условию, заданному операндом WHERE.

Новые значения полей в записях могут быть пустыми (NULL), либо вычисляться в соответствии с арифметическим выражением. Правила записи арифметических и логических выражений аналогичны соответствующим правилам оператора SELECT.

Пример 8. Изменение записей.

Пусть необходимо увеличить на 500 единиц зарплату тем служащим, которые получают не более 6000 (по таблице EMP). Запрос, сформулированный с помощью оператора SELECT, может выглядеть так:

```
UPDATE emp
    SET sal = 6500
    WHERE sal <= 6000.
```

10. Оператор ***вставки новых записей*** имеет форматы двух видов:

```
INSERT INTO <имя таблицы>
    [<список столбцов>]
    VALUES (<список значений>)
```

и

```
INSERT INTO <имя таблицы>
    [<список столбцов>]
    <предложение SELECT>
```

В первом формате оператор INSERT предназначен для ввода новых записей с заданными значениями в столбцах. Порядок перечисления имен столбцов должен соответствовать порядку значений, перечисленных в списке операнда VALUES. Если <список столбцов> опущен, то в <списке значений> должны быть перечислены все значения в порядке столбцов структуры таблицы.

Во втором формате оператор INSERT предназначен для ввода в заданную таблицу новых строк, отобранных из другой таблицы с помощью предложения SELECT.

Пример 9. Ввод записей.

Ввести в таблицу EMP запись о новом сотруднике. Для этого можно записать такой оператор вида:

```
INSERT INTO emp
```

VALUES («Ivanov», 7500, «Lee», «cosmetics»).

11. Оператор *удаления записей* имеет формат вида:

DELETE FROM <имя таблицы>
[WHERE <условие>]

Результатом выполнения оператора DELETE является удаление из указанной таблицы строк, которые удовлетворяют условию, определенному операндом WHERE. Если необязательный операнд WHERE опущен, то есть условие отбора удаляемых записей отсутствует, удалению подлежат все записи таблицы.

Пример 10. Удаление записей.

В связи с ликвидацией отдела игрушек (toy), требуется удалить из таблицы EMP всех сотрудников этого отдела. Оператор DELETE для этой задачи будет выглядеть так:

```
DELETE FROM emp  
WHERE dept = «toy».
```

В заключение отметим, что, по словам Дейта, язык SQL является гибридом реляционной алгебры и реляционного исчисления. В нем имеются элементы алгебры (оператор объединения UNION) и исчисления (квантор существования EXISTS). Кроме того, язык SQL обладает реляционной полнотой.

Контрольные вопросы и задания

1. Дайте определение реляционной модели и назовите составляющие ее элементы.
2. Охарактеризуйте составные элементы реляционной модели данных и формы их представления.
3. Приведите математическое описание понятия отношения.
4. Что такое домен отношения?
5. Дайте определение схемы отношения.
6. Что представляет собой primary ключ отношения, для чего он задается?
7. Назовите условия, при соблюдении которых таблицу можно считать отношением.
8. Что такое индекс, для чего используется индексирование?
9. Изобразите схему одноуровневой индексации и дайте ей характеристику.
10. Изобразите схему двухуровневой индексации и дайте ей характеристику.
11. Что такое вторичный индекс, в чем его отличие от первичного индекса?
12. Приведите схему возможной организации связи вторичного индекса с элементами базы данных.

13. Опишите действие механизма контроля целостности при манипулировании данными в таблицах.
14. Дайте общую характеристику теоретических языков запросов.
15. Назовите операции реляционной алгебры, предложенной Коддом, и приведите графическую интерпретацию для операций пересечения и произведения.
16. Охарактеризуйте общий и частные случаи операции соединения.
17. Назовите правила записи выражений реляционной алгебры.
18. Назовите и охарактеризуйте дополнительные операции реляционной алгебры, предложенные Дейтом.
19. Охарактеризуйте варианты реляционного исчисления.
20. Запишите выражение реляционного исчисления на кортежах, соответствующее запросу: «Получить имена поставщиков, которые поставляют все детали».
21. Охарактеризуйте язык QBE.
22. Дайте определение понятия элемента примера и приведите пример его использования в шаблоне запроса на выборку.
23. Назовите предполагаемые направления совершенствования языка QBE в современных СУБД.
24. Охарактеризуйте язык SQL.
25. Покажите, что с помощью выражения $((S [П#] MINUS (SP WHERE Д# = 'P2') [П#]) JOIN S) [Имя]$ реляционной алгебры можно получить имена поставщиков, которые не поставляют деталь P2 (рис. 3.7).
26. Сформулируйте запрос, в котором требуется определить названия фирм, которые поставляют товары, отличные от товаров, предлагаемых фирмой Ренсрафт (подраздел 3.8).

Литература

1. *Дейт К. Дж.* Введение в системы баз данных / Пер. с англ. 6-е изд. К.: Диалектика, 1998.
2. *Замулин А. В.* Системы программирования баз данных и знаний. Новосибирск.: Наука. Сиб. отд., 1990.
3. *Мартин Дж.* Организация баз данных в вычислительных системах. / Пер. с англ. М.: Мир, 1980.
4. *Романов Б. А., Кушниренко А. С.* dBase IV. Назначение, функции, применение. М.: Радио и связь, 1991.
5. *Ульман Дж.* Основы систем баз данных. М.: Финансы и статистика, 1983.

4. Информационные системы в сетях

Создание и применение информационных систем в сетях компьютеров, с одной стороны, дает заметные преимущества, с другой стороны, вызывает ряд проблем. В частности, возникают проблемы администрирования и защиты информации. В разделе рассматриваются основные понятия, связанные с сетями компьютеров и информационными системами в них, варианты архитектуры клиент-сервер, управление данными и доступ к ним, особенности информационных систем в локальных сетях, Интернете и интранете.

4.1. Основные понятия

Основные понятия сетей ЭВМ раскроем, рассматривая виды и состав сетей, используемое программное и аппаратное обеспечение, а также методы управления ресурсами.

Виды и состав сетей

Сетью называется совокупность компьютеров или рабочих станций (РС), объединенных средствами передачи данных. Средства передачи данных, в свою очередь, в общем случае могут состоять из следующих элементов: связных компьютеров, каналов связи (спутниковых, телефонных, цифровых, волоконно-оптических, радио- и других), коммутирующей аппаратуры, ретрансляторов, различного рода преобразователей сигналов и других элементов и устройств.

Современные сети можно классифицировать по различным признакам: по удаленности компьютеров, топологии, назначению, перечню предоставляемых услуг, принципам управления (централизованные и децентрализованные), методам коммутации (без коммутации, телефонная коммутация, коммутация цепей, сообщений, пакетов и дейтаграмм), видам среды передачи и т.д.

В зависимости от удаленности компьютеров сети условно разделяют на **локальные и глобальные**.

Произвольная **глобальная сеть** может включать другие глобальные сети, локальные сети, а также отдельно подключаемые к ней компьютеры (удаленные компьютеры) или отдельно подключаемые устройства ввода-вывода. Глобальные сети различают четырех основных видов: городские, региональные, национальные и транснациональные. В качестве устройств ввода-вывода могут использоваться, например, печатающие и копирующие устройства, кассовые и банковские аппараты, дисплеи (терминалы) и факсы. Перечисленные элементы сети могут быть удалены друг от друга на значительное расстояние.

Примером глобальной сети служит одна из первых в мире сеть ARPANET, а также сеть Интернет.

В **локальных вычислительных сетях** (ЛВС) компьютеры расположены на расстоянии до нескольких километров и обычно соединены при помощи скоростных линий связи со скоростью обмена от 1 до 10 и более Мбитов/с (не исключается соединение компьютеров и с помощью низкоскоростных телефонных линий). ЛВС обычно развертываются в рамках некоторой организации (корпорации, учреждения). Поэтому их иногда называют *корпоративными системами или сетями*. Компьютеры при этом, как правило, находятся в пределах одного помещения, здания или соседних зданий.

В глобальной сети основным видом взаимодействия между независимыми компьютерами является обмен сообщениями. Более интенсивный обмен информацией происходит в локальных сетях. В них, по существу, организовано управление аппаратно-программными ресурсами всех входящих в сеть компьютеров. Реализует эти функции сетевое ПО. Вкратце остановимся на программно-аппаратных ресурсах и методах управления ими в ЛВС.

Программное обеспечение ЛВС

Независимо от того, в какой сети работает некоторый компьютер, функции установленного на нем программного обеспечения условно можно разделить на две группы: *управление ресурсами* самого компьютера (в том числе и в интересах решения задач для других компьютеров) и *управление обменом* с другими компьютерами (сетевые функции).

Собственными ресурсами компьютера традиционно управляет ОС. Функции сетевого управления реализует **сетевое ПО**, которое может быть выполнено как в виде отдельных пакетов сетевых программ, так и виде сетевой ОС (СОС).

При разработке сетевого ПО используется иерархический подход, предполагающий определение совокупности сравнительно независимых уровней и интерфейсов между ними. Это позволяет легко модифицировать алгоритмы программ произвольного уровня без существенного изменения других уровней. В общем случае допускается упрощение функций некоторого уровня или даже его полная ликвидация.

Для упорядочения разработки сетевого ПО и обеспечения возможности взаимодействия любых вычислительных систем, Международная Организация по Стандартизации (International Organization for Standardization – ISO) разработала **Эталонную Модель взаимодействия открытых систем** (Open System Interconnection Reference model – модель OSI).

Эталонная Модель определяет следующие семь функциональных уровней:

- физический (physical layer);
- канальный или управления линией (звеном) передачи (data link);
- сетевой (network layer);
- транспортный (transport layer);

- сеансовый (session layer);
- представительный (presentation layer);
- прикладной или уровень приложений (application layer).

Отличия сетей друг от друга вызваны особенностями используемого аппаратного и программного обеспечения, различной интерпретацией рекомендаций фирмами-разработчиками, различием требований к системе со стороны решаемых задач (требования защищенности информации, скорости обмена, безошибочности передачи данных и т. д.) и другими причинами. В сетевом ПО локальных сетей часто наблюдается сокращение числа реализуемых уровней.

Аппаратные средства ЛВС

Основными аппаратными компонентами ЛВС являются: рабочие станции, серверы, интерфейсные платы (сетевые адаптеры) и кабели.

Рабочие станции (PC) – это, как правило, персональные ЭВМ, которые являются рабочими местами пользователей сети.

Иногда в PC, непосредственно подключенной к сетевому кабелю, могут отсутствовать накопители на магнитных дисках. Такие PC называют **бездисковыми рабочими станциями**. Основным *преимуществом* бездисковых PC является низкая стоимость, а также высокая защищенность от несанкционированного проникновения в систему пользователей и компьютерных вирусов. *Недостаток* бездисковой PC заключается в невозможности работать в автономном режиме (без подключения к серверу) и иметь собственные архивы данных и программ.

Серверы в ЛВС выполняют функции распределения сетевых ресурсов. Обычно его функции возлагают на достаточно мощный ПК, мини-ЭВМ, большую ЭВМ или специальную ЭВМ-сервер. В одной сети может быть один или несколько серверов. Каждый из серверов может быть отдельным или совмещенным с PC. В последнем случае только часть ресурсов сервера оказывается общедоступной.

При наличии в ЛВС нескольких серверов, каждый из них управляет работой подключенных к нему PC. Совокупность компьютеров сервера и относящихся к нему PC часто называют **доменом**. Иногда в одном домене находится несколько серверов. Обычно один из них является главным, а другие – выполняют роль резерва (на случай отказа главного сервера) или логического расширения основного сервера.

Сетевой адаптер может находиться на материнской плате компьютера либо быть выполнен в виде отдельного устройства (карты), подключаемого к стандартному разъему. Он имеет три основные характеристики: скорость передачи (в мегабитах в секунду), разрядность (8, 16, 32, 64) и используемый метод доступа к сетевому каналу данных. АдAPTERЫ для организации беспроводных сетей характеризуются также частотой радиосигнала и обычно подключаются к разъему PCI или USB.

Существуют различные схемы объединения компьютеров в ЛВС. Классическими считаются топологии «звезда», «кольцо» и «общая шина». Наиболее широко используются следующие стандартизованные методы доступа к сетевому каналу:

- Ethernet (поддерживает шинную топологию);
- Arcnet (поддерживает звездную топологию);
- Token-Ring (поддерживает кольцевую топологию).

Конфигурация соединения элементов в сеть (топология) во многом определяет такие важнейшие характеристики сети, как ее надежность, производительность, стоимость, защищенность и т. д.

Одним из подходов к классификации топологий ЛВС является выделение двух основных классов топологий: **широковещательных и последовательных**.

В **широковещательных** конфигурациях каждый персональный компьютер передает сигналы, которые могут быть восприняты остальными компьютерами. К таким конфигурациям относятся топологии «общая шина», «дерево», «звезда с пассивным центром». Сеть типа «звезда с пассивным центром» можно рассматривать как разновидность «дерева», имеющего корень с ответвлением к каждому подключенному устройству.

В **последовательных** конфигурациях каждый физический подуровень передает информацию одному компьютеру. Примерами последовательных конфигураций являются: произвольная (произвольное соединение компьютеров), иерархическая, «кольцо», «цепочка», «звезда с интеллектуальным центром», «снежинка».

Для соединения компьютеров в ЛВС чаще всего используют коаксиальный кабель (тонкий и толстый). Помимо коаксиального кабеля может применяться «витая пара» и оптоволокно. В последнее время ведутся интенсивные работы по разработке и внедрению беспроводных радиосетей. Известные системы на их основе, по сравнению с кабельными системами, пока несколько уступают по скорости передачи данных и дальности приема (сотни метров), но позволяют создавать мобильные распределенные системы.

К дополнительному оборудованию ЛВС относят переключатели (switch), концентраторы (hub), маршрутизаторы (router), трансиверы (transceiver), шлюзы (gateway), повторители (repeater), мосты (bridge), источники бесперебойного питания, модемы, различные разъемы (коннекторы, терминалы) и т.д.

Принципы управления

Существует два основных метода (принципа) управления в ЛВС: централизованное и децентрализованное. В централизованных сетях одна или несколько ПЭВМ являются центральными, а остальные — рабочими станциями (PC).

Центральный узел сети, часто называемый компьютером-сервером (КС), может находиться на отдельном компьютере или совмещаться с РС. В первом случае говорят о выделенном КС, а во втором — о совмещенном КС.

Основное назначение КС — управление передачей данных в сети и хранение файлов, используемых многими РС. Исходя из этого, под КС обычно выделяют наиболее производительную и имеющую значительную память ПЭВМ. Кроме того, к КС обычно подключается дорогостоящее оборудование (лазерные принтеры, факсы, модемы, сканеры и т. д.).

Существует множество сетевых ОС, реализующих централизованное управление. Среди них Microsoft Windows NT Server, Novell NetWare (версии 3.X и 4.X), Microsoft Lan Manager, OS/2 Warp Server Advanced, VINES 6.0 и другие.

Преимуществом централизованных сетей является высокая защищенность сетевых ресурсов от несанкционированного доступа, удобство администрирования сети, возможность создания сетей с большим числом узлов. Основной недостаток состоит в уязвимости системы при нарушении работоспособности файл-сервера (это преодолевается при наличии нескольких серверов или принятии других мер), а также в предъявлении довольно высоких требований к ресурсам серверов.

Сети с *децентрализованным управлением* (одноранговые сети) не содержат КС. В них функции управления сетью в соответствии с некоторой дисциплиной поочередно передаются от одной РС к другой. Децентрализованное управление, как правило, применяется в сетях со слабыми компьютерами и простейшими обменами между ними на уровне файлов, а также без серьезного контроля прав доступа к ресурсам сети. Основные ресурсы всех РС обычно оказываются общедоступными, хотя система не всегда обеспечивает корректное их совместное использование на прикладном уровне.

Наиболее распространенными программными продуктами, позволяющими строить одноранговые сети, являются следующие программы и пакеты: Novell NetWare Lite, Windows for Workgroups, Windows 95/98, Artisoft LANtastic, LANsmart, Invisible Software NET-30 и другие.

Развертывание одноранговой сети для небольшого числа РС часто позволяет построить более эффективную и живучую распределенную вычислительную среду. Сетевое программное обеспечение в них является более простым по сравнению с централизованными сетями. Здесь не требуется установка файл-сервера (компьютера и соответствующих программ), что существенно удешевляет систему. Однако такие сети слабее с точки зрения защиты информации и администрирования.

Говоря о сетях, часто используют термины «сервер» и «клиент». Любое взаимодействие в сети предполагает как минимум два элемента: потребляющий и предоставляющий ресурсы (сервис или услуги). Потребитель ресурсов называется клиентом, а предоставляющий ресурсы компонент — сервером.

ром (см. раздел 1). Основными видами ресурсов являются следующие: аппаратные (целая ЭВМ, дисковый накопитель, устройство печати и т. д.), программные и информационные.

Если в качестве ресурса рассматривается вся ЭВМ, то говорят о компьютере-клиенте и компьютере-сервере. Если подразумевается некоторый аппаратный ресурс, то используют такие термины как: диск-сервер (файловый сервер или файл-сервер), сервер печати. Типичными клиентами в сетях являются: ЭВМ, пользователь или программа.

Возможно уточнение назначения компьютера при обработке информации. Тогда говорят о серверах сообщений (обработка поступающих сообщений), серверах БД (обработка запросов к БД), серверах приложений (выполнение приложений пользователя) и т. д.

Иногда одним термином называют разные (аппаратные, программные или аппаратно-программные) компоненты вычислительной системы. Так, сервером печати может служить компьютер с подключенным к нему принтером, программа печати или компьютер с программным обеспечением управления печатью.

4.2. Модели архитектуры клиент-сервер

При построении распределенных ИС, работающих с БД, широко используется архитектура клиент-сервер. Ее основу составляют принципы организации взаимодействия клиента и сервера при управлении БД. Один из вариантов архитектуры клиент-сервер рассмотрен в подразделе 1.2.

Чтобы охарактеризовать основные схемы взаимодействия процессов управления БД, воспользуемся Эталонной моделью Архитектуры открытых систем OSI. Согласно этой модели, функция управления БД относится к прикладному уровню.

Остановимся на двух самых верхних уровнях: *прикладном* и *представительном*, которые в наибольшей степени являются предметом внимания со стороны разработчика и пользователя. Остальные функции будем считать связанными функциями, необходимыми для реализации двух первых. При этом будем придерживаться широкого толкования термина СУБД, понимая под ним все программные системы, которые используют информацию из БД.

Как поддерживающая интерфейс с пользователем программа, СУБД, в общем случае, реализует следующие основные функции:

- управление данными, находящимися в базе;
- обработка информации с помощью прикладных программ;
- представление информации в удобном для пользователя виде.

Если система размещается на одной ЭВМ, то все функции собраны в одной программе и вызываются по схеме, подобной рассмотренной в разделе 1.

При размещении СУБД в сети возможны различные варианты распределения функций по узлам. В зависимости от числа узлов сети, между которыми выполняется распределение функций СУБД, можно выделить *двузвенные* модели, *трехзвенные* модели и т. д. Место разрыва функций соединяется коммуникационными функциями (средой передачи информации в сети).

Двузвенные модели распределения функций

Двузвенные модели соответствуют распределению функций СУБД между двумя узлами сети. Компьютер (узел сети), на котором обязательно присутствует функция управления данными, назовем *компьютером-сервером*. Компьютер, близкий к пользователю и обязательно занимающийся вопросами представления информации, назовем *компьютером-клиентом*.

Наиболее типичными вариантами (рис. 4.1) разделения функций между компьютером-сервером и компьютером-клиентом являются следующие:

- распределенное представление;
- удаленное представление;
- распределенная функция;
- удаленный доступ к данным;
- распределенная БД.

Перечисленные способы распределения функций в системах с архитектурой клиент-сервер иллюстрируют различные варианты: от мощного сервера, когда практически вся работа производится на нем, до мощного клиента, ког-

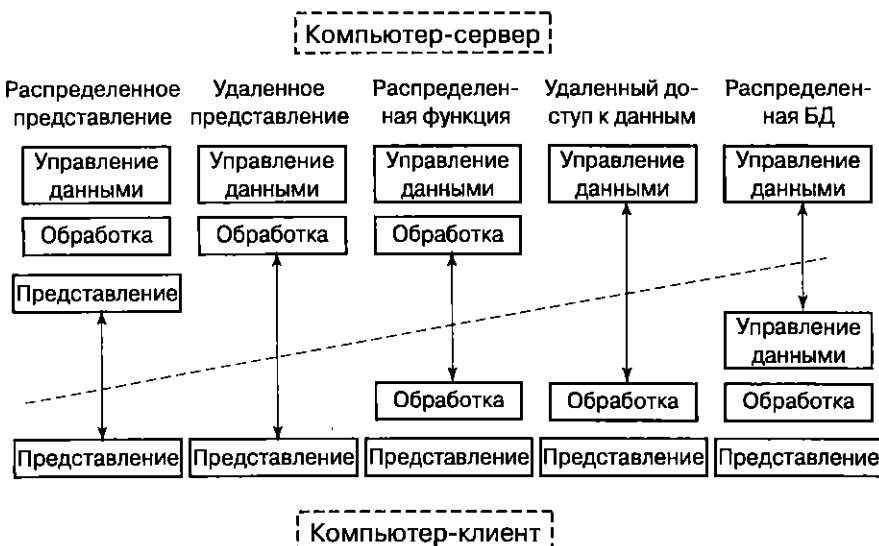


Рис. 4.1. Спектр моделей архитектуры клиент-сервер

да большая часть функций выполняется на рабочей станции, а сервер обрабатывает поступающие к нему по сети SQL-вызовы.

В моделях удаленного доступа к данным и удаленного представления производится строгое распределение функций между компьютером-клиентом и компьютером-сервером. В других моделях имеет место выполнение одной из следующих функций одновременно на двух компьютерах: управления данными (модель распределенной БД), обработки информации (модель распределенной функции), представления информации (модель распределенного представления).

Рассмотрим сначала модели *удаленного доступа к данным и удаленного представления (сервера БД)* как наиболее широко распространенные.

В модели *удаленного доступа к данным* (Remote Data Access – RDA) программы, реализующие функции представления информации и логику прикладной обработки, совмещены и выполняются на компьютере-клиенте. Обращение за сервисом управления данными происходит через среду передачи с помощью операторов языка SQL или вызовом функций специальной библиотеки API (Application Programming Interface – интерфейса прикладного программирования).

Основное достоинство RDA-модели состоит в большом обилии готовых СУБД, имеющих SQL-интерфейсы, и существующих инструментальных средств, обеспечивающих быстрое создание программ клиентской части. Средства разработки чаще всего поддерживают графический интерфейс пользователя в MS Windows, стандарт интерфейса ODBC и средства автоматической генерации кода. Подавляющее большинство средств разработки использует языки четвертого поколения.

Недостатками RDA-модели являются, во-первых, довольно высокая загрузка системы передачи данных вследствие того, что вся логика сосредоточена в приложении, а обрабатываемые данные расположены на удаленном узле. Как увидим далее, во время работы приложений обычно по сети передаются целые БД.

Во-вторых, системы, построенные на основе модели RDA, неудобны с точки зрения разработки, модификации и сопровождения. Основная причина состоит в том, что в получаемых приложениях прикладные функции и функции представления тесно взаимосвязаны. Поэтому даже при незначительном изменении функций системы требуется переделка всей прикладной ее части, усложняющая разработку и модификацию системы.

Модель *сервера БД* (DataBase Server – DBS) отличается от предыдущей модели тем, что функции компьютера-клиента ограничиваются функциями представления информации, в то время как прикладные функции обеспечиваются приложением, находящимся на компьютере-сервере. Эта модель является более технологичной чем RDA-модель и применяется в таких СУБД, как Ingress, Sybase и Oracle. При этом приложения реализуются в виде *хранимых процедур*.

Процедуры обычно хранятся в словаре БД и разделяются несколькими клиентами. В общем случае хранимые процедуры могут выполняться в режимах компиляции и интерпретации.

Достоинствами модели DBS являются: возможность хорошего централизованного администрирования приложений на этапах разработки, сопровождения и модификации, а также эффективное использование вычислительных и коммуникационных ресурсов. Последнее достигается за счет того, что выполнение программ в режиме коллективного пользования требует существенно меньших затрат на пересылку данных в сети.

Один из *недостатков* модели DBS связан с ограничениями средств разработки хранимых процедур. Основное ограничение — сильная привязка операторов хранимых процедур к конкретной СУБД. Язык написания хранимых процедур, по сути, является процедурным расширением языка SQL, и не может соперничать по выразительным средствам и функциональным возможностям с традиционными языками третьего поколения, такими как С и Pascal. Кроме того, в большинстве СУБД нет удовлетворительных средств отладки и тестирования хранимых процедур, что делает их механизм опасным инструментом — неотлаженные программы могут приводить к некорректностям БД, зависаниям серверных и клиентских программ во время работы системы и т. п.

Другим *недостатком* DBS-модели является низкая эффективность использования вычислительных ресурсов ЭВМ, поскольку не удается организовать управление входным потоком запросов к программам компьютера-сервера, а также обеспечить перемещение процедур на другие компьютеры-серверы.

В модели *распределенного представления* имеется мощный компьютер-сервер, а клиентская часть системы практически вырождена. Функцией клиентской части является просто отображение информации на экране монитора и связь с основным компьютером через локальную сеть.

СУБД подобного рода могут иметь место в сетях, поддерживающих работу так называемых *X-терминалов*. В них основной компьютер (хост-машина) должен иметь достаточную мощность, чтобы обслуживать несколько X-терминалов. X-терминал тоже должен обладать достаточно быстрым процессором и иметь достаточный объем оперативной памяти (дисковые накопители отсутствуют). Часто X-терминалы создают на базе RISC-компьютеров (*restricted [reduced] instruction set computer*) — компьютеров с сокращенным набором команд. Все программное обеспечение находится на хост-машине. Программное обеспечение X-терминала, выполняющее функции управления представлением и сетевые функции, загружается по сети с сервера при включении X-терминала.

Модель распределенного представления имели СУБД ранних поколений, которые работали на малых, средних и больших ЭВМ. В роли X-терминалов выступали дисплейные станции и абонентские пункты (локальные и удаленные).

ные). В этом случае основную часть функций представления информации реализовывали сами СУБД, а окончательное построение изображений на терминалах пользователя выполнялось на оконечных устройствах.

По модели распределенного представления построены системы обслуживания пользователей БД в гетерогенной (неоднородной) среде. Серверная часть таких систем обычно обеспечивает некоторый унифицированный интерфейс, а клиентские части реализуют функции учета специфики оконечного оборудования или преобразования одного формата представления информации в другой.

Модель распределенного представления реализует централизованную схему управления вычислительными ресурсами. Отсюда следуют ее основные достоинства — простота обслуживания и управления доступом к системе и относительная дешевизна (ввиду невысокой стоимости оконечных терминалов). Недостатками модели являются уязвимость системы при невысокой надежности центрального узла, а также высокие требования к серверу по производительности при большом числе клиентов.

В модели **распределенной функции** логика обработки данных распределена по двум узлам. Такую модель могут иметь ИС, в которых общая часть прикладных функций реализована на компьютере-сервере, а специфические функции обработки информации выполняются на компьютере-клиенте. Функции общего характера могут включать в себя стандартное обеспечение целостности данных, например, в виде хранимых процедур, а оставшиеся прикладные функции реализуют специальную прикладную обработку. Подобную модель имеют также ИС, использующие информацию из нескольких неоднородных БД.

Модель **распределенной БД** предполагает использование мощного компьютера-клиента, причем данные хранятся на компьютере-клиенте и на компьютере-сервере. Взаимосвязь обеих баз данных может быть двух разновидностей: а) в локальной и удаленной базах хранятся отдельные части единой БД; б) локальная и удаленная БД являются синхронизируемыми друг с другом копиями.

Достоинством модели распределенной БД является гибкость создаваемых на ее основе ИС, позволяющих компьютеру-клиенту обрабатывать локальные и удаленные БД. При наличии механизмов координации соответствия копий система в целом, кроме того, обладает высокой живучестью, так как разрыв соединения клиента и сервера не приводит к краху системы, а ее работа может быть восстановлена с возобновлением соединения. К недостатку модели можно отнести высокие затраты при выполнении большого числа одинаковых приложений на компьютерах-клиентах.

Трехзвенная модель распределения функций

Трехзвенная модель распределения функций представляет собой типовой вариант, при котором каждая из трех функций приложения реализуется

на отдельном компьютере. Варианты распределения функций приложения на большее число компьютеров могут иметь место, но ввиду их редкого применения рассматриваться не будут.

Рассматриваемая нами модель имеет название **модель сервера приложений**, или **AS-модель** (Application Server), и показана на рис. 4.2.



Рис. 4.2. Трехзвенная модель сервера приложений

Согласно трехзвенной AS-модели, отвечающий за организацию диалога с конечным пользователем процесс, как обычно, реализует функции представления информации и взаимодействует с компонентом приложения так же, как в модели DBS. Компонент приложения, располагаясь на отдельном компьютере, в свою очередь, связан с компонентом управления данными подобно модели RDA.

Центральным звеном AS-модели является сервер приложений. На сервере приложений реализуется несколько прикладных функций, каждая из которых оформлена как служба предоставления услуг всем требующим этого программам. Серверов приложений может быть несколько, причем каждый из них предоставляет свой вид сервиса. Любая программа, запрашивающая услугу у сервера приложений, является для него клиентом. Поступающие от клиентов к серверам запросы помещаются в очередь, из которой выбираются в соответствии с некоторой дисциплиной, например, по приоритетам.

Компонент, реализующий функции представления и являющийся клиентом для сервера приложений, в этой модели трактуется более широко, чем обычно. Он может служить для организации интерфейса с конечным пользователем, обеспечивать прием данных от устройств, например, датчиков, или быть произвольной программой.

Достоинством AS-модели является гибкость и универсальность вследствие разделения функций приложения на три независимые составляющие. Во многих случаях эта модель оказывается более эффективной по сравнению с двухзвенными. **Основной недостаток** модели — более высокие затраты ресурсов

компьютеров на обмен информацией между компонентами приложения по сравнению с двухзвенными моделями.

Примерами программных продуктов, реализующих среду функционирования приложений на компьютерах-серверах приложений, являются BEA WebLogic Server (BEA Systems Corp.), Inprise Application Server (Inprise Corp.) и IBM WebSphere Application Server (IBM Corp.).

Сложные схемы взаимодействия

Возможны более сложные схемы взаимодействия, например, схемы, в которых элемент, являющийся сервером для некоторого клиента, в свою очередь, выступает в роли клиента по отношению к другому серверу (рис. 4.3). Пример этого мы наблюдали в AS-модели.



Рис. 4.3. Цепочка взаимодействий типа клиент-сервер

Возможно также, что в распределенной вычислительной системе при работе с БД имеются множественные связи (статические), когда один объект по отношению к одним является клиентом, а по отношению к другим — сервером (рис. 4.4).

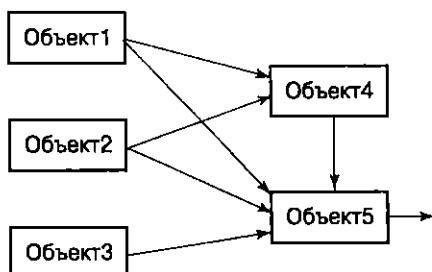


Рис. 4.4. Множественные связи взаимодействия типа клиент-сервер

При рассмотрении взаимодействия объектов в *динамике* получаются еще более сложные схемы взаимодействия. Примером такой схемы является случай, когда в процессе работы роли объектов меняются: объект, являющийся в некоторый момент времени клиентом по отношению к другому объекту, в последующем становится сервером для другого объекта.

Модель монитора транзакций

Как отмечалось, наиболее гибкой и универсальной моделью распределения функций СУБД является AS-модель. Она описывает взаимодействие трех основных элементов: клиента, сервера приложений и сервера БД, но не затрагивает вопросы организации функционирования программного обеспечения при обработке информации, в частности при выполнении транзакций. Для преодоления этого недостатка предложена модель *монитора транзакций*.

Мониторы обработки транзакций (Transaction Processing Monitor — TPM), или **мониторы транзакций**, представляют собой программные системы категории промежуточного слоя (Middleware), обеспечивающие эффективное управление информационно-вычислительными ресурсами в распределенной вычислительной системе. Основное их назначение — организация гибкой, открытой среды для разработки и управления мобильными приложениями, оперативно обрабатывающими распределенные транзакции. Применение мониторов транзакций наиболее эффективно в гетерогенных вычислительных системах.

Приложение TPM позволяет выполнять масштабирование системы, поддерживать функциональную полноту и целостность приложений а также повысить производительность обработки данных при невысокой стоимости на-кладных расходов.

Принципы организации обработки информации с помощью монитора транзакций описываются **моделью монитора транзакций** X/Open DTP (Distributed Transaction Processing — обработка распределенных транзакций). Эта модель (рис. 4.5) включает в себя три объекта: прикладную программу, менеджер ресурсов (Resource Manager — RM) и монитор, или менеджер транзакций (Transaction Manager — TM).

В качестве прикладной программы может выступать произвольная программа-клиент. RM выполняет функции сервера ресурса некоторого вида.

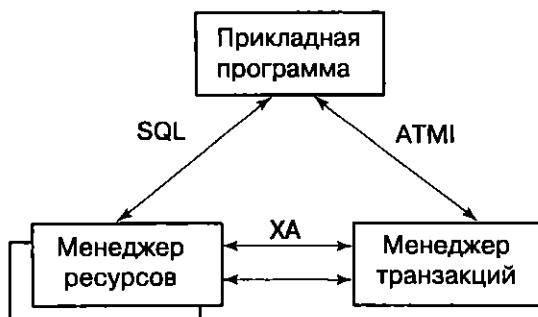


Рис. 4.5. Модель обработки транзакций X/Open DTP

Прикладная программа взаимодействует с ТРМ с помощью набора специальных функций либо посредством операторов SQL (когда сервером является сервер БД).

Интерфейс ATMI (Application Transaction Monitor Interface – интерфейс монитора транзакций приложения) позволяет вызывать функции ТРМ на некотором языке программирования, например С.

Функции менеджера ресурсов обычно выполняют серверы БД или СУБД. В задачах организации управления обработкой распределенных транзакций (транзакций, затрагивающих программные объекты вычислительной сети) ТМ взаимодействует с ТРМ, который должен поддерживать протокол двухфазной фиксации транзакций (см. подр. 4.3) и удовлетворять стандарту X/Open XA. Примерами СУБД, поддерживающих протокол двухфазной фиксации транзакций, являются Oracle 7.0, Open INGRES и Informix-Online 5.0.

Понятие транзакции в ТРМ несколько шире, чем в СУБД, где транзакция включает в себя элементарные действия над данными базы. Здесь транзакция может охватывать и другие необходимые действия: передачу сообщения, запись информации в файл, опрос датчиков и т. д. Это значит, что ТРМ предоставляет более мощные средства управления вычислительным процессом. Транзакции, которые поддерживают ТРМ, называют также *прикладными* или *бизнес-транзакциями*.

Модель X/Open DTP не раскрывает структуру ТМ в деталях, а определяет состав компонентов распределенной системы обработки информации и как эти компоненты взаимодействуют друг с другом. Практические реализации этой модели, естественно, могут отличаться друг от друга. В числе примеров реализаций мониторов транзакций можно назвать ACMS, CICS, и TUXEDO System.

Для *разработчиков приложений* мониторы обработки транзакций ТРМ предоставляют удобства, связанные с возможностью декомпозиции приложений по нескольким функциональным уровням со стандартными интерфейсами, что позволяет создавать легко модифицируемые ИС со стройной архитектурой.

Прикладные программы становятся практически независимыми от конкретной реализации интерфейса с пользователем и от менеджера ресурсов. Первое означает, что для реализации функций представления можно выбрать любое удобное и привычное для разработчика средство (от языка С до какой-либо CASE-системы). Независимость от менеджера ресурсов подразумевает возможность легкой замены одного менеджера ресурсов на другой, лишь бы они поддерживали стандарт взаимодействия с прикладной программой (для СУБД – язык SQL).

Если ТРМ поддерживает множество аппаратно-программных платформ, как, например, TUXEDO System, то разрабатываемые приложения становятся, кроме того, мобильными.

Сосредоточение всех прикладных функций в серверах приложений и наличие богатых возможностей управления и администрирования существенно упрощает обновление прикладных функций (бизнес-функций) и контроль за их непротиворечивостью. Изменения в прикладных функциях при этом никак не отражаются на программах-клиентах.

Для *пользователей распределенных систем* ТРМ позволяют улучшить показатели пропускной способности и времени отклика, снизить стоимость обработки данных в оперативном режиме на основе эффективной организации вычислительного процесса.

Улучшение показателей функционирования достигается благодаря осуществлению статической и динамической балансировки нагрузки. Управление загрузкой состоит в запуске или остановке AS-процессов (программных компонентов AS-модели) в зависимости от заранее установленных параметров и текущего состояния системы. При необходимости ТРМ может тиражировать копии AS-процессов на этом или других узлах сети.

Администраторы распределенных систем, имея ТРМ, получают возможность легкого масштабирования ИС и увеличения производительности обработки информации. Здесь, кроме вертикального и горизонтального масштабирования, можно обеспечить так называемое *матричное масштабирование*. Суть его — введение дополнительных ресурсов в любую точку гетерогенной вычислительной среды без изменения архитектуры приложения, выполняемого в новой среде. Это означает, что без остановки серверов приложений в любое время может быть добавлен, например, компьютер или менеджер ресурсов.

Кроме того, администраторы систем получают возможность снизить общую стоимость программного обеспечения систем клиент-сервер. Снижение стоимости можно достигнуть простым уменьшением количества подключений к серверам БД.

Стоимость серверов БД (или СУБД) в сильной степени зависит от числа одновременных подключений к программе. Уменьшение количества подключений к серверу БД достигается путем мультиплексирования запросов или, что то же самое, логического преобразования потока запросов от многих источников к потоку от одного или нескольких источников. Выигрыш в стоимости и производительности при эффективной реализации самих ТРМ может оказаться весьма существенным.

4.3. Управление распределенными данными

С управлением данными в распределенных системах связаны следующие две группы проблем: поддержка соответствия БД вносимым изменениям и обеспечение совместного доступа нескольких пользователей к общим данным.

Проблема совместного доступа к распределенным данным, в свою очередь, тесно связана с тупиками. Одним из средств избежания тупиков являются протоколы фиксации транзакций.

Поддержка соответствия БД вносимым изменениям

В современных распределенных системах информация может храниться централизованно или децентрализованно. В первом случае проблемы идентичности представления информации для всех пользователей не существует, так как все последние изменения хранятся в одном месте. На практике чаще информация изменяется одновременно в нескольких узлах распределенной вычислительной системы. В этом случае возникает проблема контроля за всеми изменениями информации и предоставления ее в достоверном виде всем пользователям.

Существуют две основные технологии децентрализованного управления БД: *распределенных БД* (Distributed Database) и *тиражирования, или репликации, БД* (Data Replication).

Распределенная БД состоит из нескольких фрагментов, размещенных на разных узлах сети и, возможно, управляемых разными СУБД. С точки зрения программ и пользователей, обращающихся к распределенной БД, последняя воспринимается как единая локальная БД (рис. 4.6).

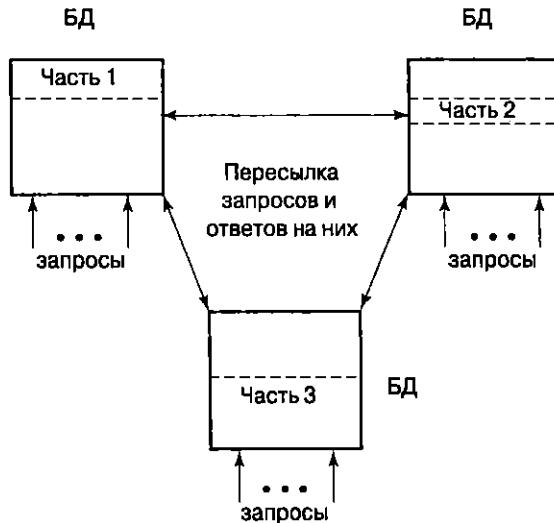


Рис. 4.6. Модель распределенной БД

Информация о местоположении каждой из частей распределенной БД и другая служебная информация хранится в так называемом *глобальном словаре*.

варе данных. В общем случае этот словарь может храниться на одном из узлов или тоже быть распределенным.

Для обеспечения корректного доступа к распределенной БД в современных системах чаще всего применяется протокол (метод) *двухфазной фиксации транзакций* (two-phase commit). Суть этого метода состоит в двухэтапной синхронизации выполняемых изменений на всех задействованных узлах. На первом этапе в узлах сети производятся изменения (пока обратимые) в их БД, о чём посылаются уведомления компоненту системы, управляющему обработкой распределенных транзакций.

На втором этапе, получив от всех узлов сообщения о правильности выполнения операций (что свидетельствует об отсутствии сбоев и отказов аппаратно-программного обеспечения), управляющий компонент выдает всем узлам команду фиксации изменений. После этого транзакция считается завершенной, а ее результат необратимым.

Основным достоинством модели распределенной БД является то, что пользователи всех узлов (при исправных коммуникационных средствах) получают информацию с учетом всех последних изменений. Второе достоинство состоит в экономном использовании внешней памяти компьютеров, что позволяет организовывать БД больших объемов.

К недостаткам модели распределенной БД относится следующее: жесткие требования к производительности и надежности каналов связи, а также большие затраты коммуникационных и вычислительных ресурсов из-за их связывания на все время выполнения транзакций. При интенсивных обращениях к распределенной БД, большом числе взаимодействующих узлов, низкоскоростных и ненадежных каналах связи обработка запросов по этой схеме становится практически невозможной.

Модель *тиражирования данных*, в отличие от технологии распределенных БД, предполагает дублирование данных (создание точных копий) в узлах сети (рис. 4.7). Данные всегда обрабатываются как обычные локальные. Поддержку идентичности копий друг другу в асинхронном режиме обеспечивает компонент системы, называемый *репликатором* (replicator). При этом между узлами сети могут передаваться как отдельные изменения, так и группы изменений. В течение некоторого времени копии БД могут отличаться друг от друга.

К основным достоинствам модели тиражирования БД (в сравнении с предыдущей моделью) относятся: более высокая скорость доступа к данным, так как они всегда есть в узле; существенное уменьшение передаваемого по каналам связи потока информации, поскольку происходит передача не всех операций доступа к данным, а только изменений в БД; повышение надежности механизмов доступа к распределенным данным, поскольку нарушение связи не приводит к потере работоспособности системы (предполагается буферизация потока изменений, позволяющая корректно возобновить работу после восстановления связи).

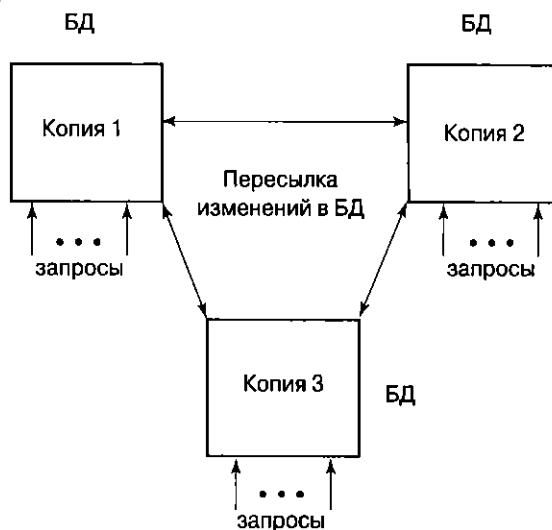


Рис. 4.7. Модель тиражирования БД

Основной недостаток модели тиражирования БД заключается в том, что на некотором интервале времени возможно «расхождение» копий БД. Если отмеченный недостаток некритичен для прикладных задач, то предпочтительно иметь схему с тиражированием БД.

Доступ к общим данным

При обслуживании обращений к общим данным средства управления БД должны обеспечивать по крайней мере два основных метода доступа: монопольный и коллективный. Основными объектами доступа в различных системах могут быть целиком БД, отдельные таблицы, записи, поля записей. В СУБД, предоставляющих возможность разработки, объектами доступа также могут выступать спецификации отчетов и экранных форм, запросы и программы.

Монопольный доступ обычно используется в двух случаях:

- во-первых, когда требуется исключить доступ к объектам со стороны других пользователей (например, при работе с конфиденциальной информацией);
- во-вторых, когда производятся *ответственные операции* с БД, не допускающие других действий, например, изменение структуры БД.

В первом случае пользователь с помощью диалоговых средств СУБД или прикладной программы устанавливает явную блокировку. Во втором случае пользователь тоже может установить явную блокировку, либо положиться на СУБД. Последняя обычно автоматически устанавливает неявную (без ведома пользователя или приложения) блокировку, если это необходимо.

В режиме **коллективного доступа** полная блокировка на используемые объекты, как правило, не устанавливается. Коллективный доступ возможен,

например, при одновременном просмотре таблиц. Попытки получить монопольный доступ к объектам коллективного доступа должны быть пресечены. Например, в ситуации когда один или несколько пользователей просматривают таблицу, а другой пользователь собирается удалить эту же таблицу.

Для организации коллективного доступа в СУБД применяется **механизм блокировок**. Суть блокировки состоит в том, что на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других потребителей временно запрещается или ограничивается. Например, при копировании таблицы она блокируется от изменения, хотя и разрешено просматривать ее содержимое.

Рассмотрим некоторый типичный набор блокировок. В конкретных программах схемы блокирования объектов могут отличаться от описываемой. Выделим четыре вида блокировок, перечисленные в порядке убывания строгости ограничений на возможные действия:

- полная блокировка;
- блокировка от записи;
- предохраниющая блокировка от записи;
- предохраниющая полная блокировка.

Полная блокировка. Означает полное запрещение всяких операций над основными объектами (таблицами, отчетами и экранными формами). Этот вид блокировки обычно применяется при изменении структуры таблицы.

Блокировка от записи. Накладывается в случаях, когда можно использовать таблицу, но без изменения ее структуры или содержимого. Такая блокировка применяется, например, при выполнении операции слияния данных из двух таблиц.

Предохраниющая блокировка от записи. Предохраняет объект от наложения на него со стороны других операций полной блокировки, либо блокировки от записи. Этот вид блокировки позволяет тому, кто раньше «захватил» объект, успешно завершить модификацию объекта. Предохраниющая блокировка от записи совместима с аналогичной блокировкой (предохраниющей блокировкой от записи), а также с предохраниющей полной блокировкой (см. далее). Примером необходимости использования этой блокировки является режим совместного редактирования таблицы несколькими пользователями.

Предохраниющая полная блокировка. Предохраняет объект от наложения на него со стороны других операций только полной блокировки. Обеспечивает максимальный уровень совместного использования объектов. Такая блокировка может использоваться, например, для обеспечения одновременного просмотра несколькими пользователями одной таблицы. В группе пользователей, работающих с одной таблицей, эта блокировка не позволит никому изменить структуру общей таблицы.

При незавершенной операции с некоторым объектом и запросе на выполнение новой операции с этим же объектом производится попытка эти опера-

ции совместить. Совмещение возможно тогда, когда совместимыми оказываются блокировки, накладываемые конкурирующими операциями.

В отношении перечисленных выше четырех блокировок действуют следующие правила совмещения:

- при наличии полной блокировки над объектом нельзя производить операции, приводящие хотя бы к одному из видов блокировок (полная блокировка несовместима ни с какой другой блокировкой);
- блокировка от записи совместима с аналогичной блокировкой и предохраняющей полной блокировкой;
- предохраняющая блокировка от записи совместима с обоими видами предохраняющих блокировок;
- предохраняющая полная блокировка совместима со всеми блокировками, кроме полной.

Обычно в СУБД каждой из выполняемых с БД операций соответствует определенный вид блокировки, которую эта операция накладывает на объект. Пользователям современных СУБД, работающим в интерактивном режиме, не нужно помнить все тонкости механизма блокировки, поскольку система достаточно «разумно» осуществляет автоматическое блокирование во всех случаях, когда это требуется. При этом система сама стремится предоставить всем пользователям наиболее свободный доступ к объектам. При необходимости пользователь и программист могут воспользоваться командными или языковыми средствами явного определения блокировок. Например, в СУБД Paradox для явного блокирования отдельной записи во время редактирования таблицы используется команда Record | Lock.

Тупики

Если не управлять доступом к совместно используемым объектам, то между потребителями ресурсов могут возникать тупиковые ситуации (клиинчи, «смертельные объятия» или блокировки). Следует отличать понятие блокировки в смысле контроля доступа к объектам (мы придерживаемся такого термина) от блокировки в смысле тупикового события.

Существует два основных вида тупиков: *взаимные* (deadlock) и *односторонние* (livelock).

Простейшим случаем *взаимного тупика* является ситуация, когда каждый из двух пользователей стремится захватить данные, уже захваченные другим пользователем (рис. 4.8а). В этой ситуации пользователь-1 ждет освобождения ресурса N, в то время как пользователь-2 ожидает освобождения от захвата ресурса M. Следовательно, никто из них не может продолжить работу.

В действительности могут возникать и более сложные ситуации, когда выполняются обращения трех и более пользователей к нескольким ресурсам. Пример одной из таких ситуаций приведен на рис. 4.8б.

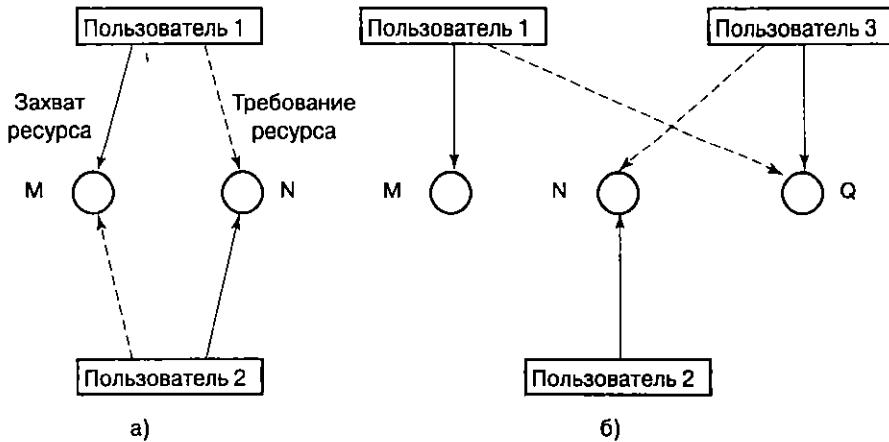


Рис. 4.8. Примеры взаимных тупиков в распределенных БД

Односторонний тупик возникает в случае требования получить моно-
польный доступ к некоторому ресурсу как только он станет доступным и не-
возможности удовлетворить это требование.

Системы управления распределенными БД, очевидно, должны иметь со-
ответствующие средства обнаружения или предотвращения конфликтов, а
также разрешения возникающих конфликтов. Одной из наиболее сложных
является задача устранения конфликтов в неоднородных системах в случае,
если некоторая программа не обрабатывает или обрабатывает некорректно
сигналы (уведомления) о наличии конфликтов. При этом важно не только
сохранить целостность и достоверность данных в распределенных БД, но и
восстановить вычислительный процесс, иногда парализующий пользовате-
лей и программы ожиданием чего-то.

Пользователи и разработчики приложений в распределенной среде долж-
ны предусматривать обработку сигналов о тупиках.

Протоколы фиксации транзакций

В настоящее время наиболее широко используются два протокола фикса-
ции транзакций: двухфазный и трехфазный. Рассмотрим их вкратце.

Прежде всего, будем считать, что выполняется распределенная транзак-
ция, с которой связан некоторый узел, функционирующий как координатор
(диспетчер). Обычно координатором является узел сети, где транзакция была
инициирована. Узлы, на которых глобальная транзакция создает агентов, на-
зовем участниками (диспетчерами ресурсов). Координатор транзакции знает
идентификаторы всех участников, а каждый участник знает идентификатор
координатора, но не обязан знать идентификаторы других участников.

Двухфазный протокол фиксации транзакций выполняется в два этапа: голосование (voting phase) и принятие решения (decision phase). Этот протокол предполагает, что каждый узел имеет свой собственный журнал, с помощью которого можно зафиксировать или отменить транзакцию. Для этого в журналы координатора и участников вносятся записи о приеме и посыпке команд (сообщений). Во избежание блокирования из-за бесконечного ожидания ответов от других участников протокол в протоколе используется механизм тайм-аутов.

На первом этапе координатор опрашивает всех участников командой PREPARE, готовы ли они к фиксации транзакции и переходит к ожиданию ответов.

На втором этапе если хотя бы один из участников потребует отката (команда ABORT) или не ответит на запрос в течение установленного интервала времени, координатор указывает всем участникам на необходимость выполнения отката данной транзакции (команда GLOBAL_ABORT). В случае получения подтверждений о фиксации транзакций от всех участников (команды READY_COMMIT), координатор отправляет всем участникам команду глобальной фиксации транзакции GLOBAL_COMMIT. Если участник не получает от координатора в установленное время команды GLOBAL_COMMIT или GLOBAL_ABORT, то он просто выполняет откат транзакции.

Когда некоторый участник требует отката транзакции, то он имеет право выполнить его немедленно. По существу каждый участник имеет право выполнить откат своей субтранзакции в любое время до отправки согласия на ее фиксацию. Такой тип отката называют *односторонним откатом*.

Действия, которые выполняются в случае возникновения тайм-аутов, описываются *протоколом аварийного завершения* (termination protocol). Предпринимаемые при этом действия зависят от того, у кого возникло это событие (у координатора или участника), в каком состоянии был объект, а также какое сообщение было получено. Например, в случае появления тайм-аута на стороне координатора во время ожидания поступления от всех участников уведомлений о принятом глобальном решении по некоторой транзакции, координатор просто повторяет рассылку о принятом решении на те узлы, которые не прислали ожидаемого подтверждения.

Действия, которые выполняются отказавшими узлами после перезапуска, описываются *протоколом восстановления*. Предпринимаемые действия системы после отказа также зависят от обстоятельств, и, главным образом, от этапа обработки транзакции координатором или участником. Пусть, например, отказ произошел у участника в его начальном состоянии (до момента получения им команды PREPARE). Поскольку координатор не может принять решение о глобальной фиксации транзакции в случае, когда хотя бы один узел не отвечает, поэтому для этого участника целесообразно выполнить откат транзакции в одностороннем порядке.

Протокол двухфазной фиксации транзакций позволяет управлять процессом обработки распределенных транзакций в достаточно широком диапазоне реальных ситуаций. Так, в случае отказа основного управляющего объекта-координатора, у участника имеется возможность обратиться за информацией о принятом глобальном решении к другим участникам взаимодействия. Если же отказал только узел координатора, а все остальные узлы «живы», то участники могут выбрать на роль координатора один из имеющихся узлов. Эти процедуры описываются *протоколами проведения выборов*. Одним из простых и эффективных протоколов проведения выборов является протокол, при котором все узлы упорядочены и имеют идентификаторы, а роль координатора выполняет узел с наименьшим идентификационным номером из числа функционирующих в произвольный момент времени. Согласно протоколу, каждый узел должен послать сообщения на узлы с большими порядковыми номерами. Тогда координатором будет тот узел, который не получит сообщений от узлов с меньшими, чем у него порядковыми номерами.

Существует несколько способов обмена сообщениями при реализации протокола двухфазной фиксации транзакций: централизованный (через узел-координатор), линейный (каждый узел взаимодействует только с узлами, имеющими ближайшие меньше и большие номера) и распределенный (сообщения пересыпаются по схеме «каждый — каждому»). Эти способы имеют свои достоинства и недостатки, которые влияют на живучесть системы, а также количество рассылаемых сообщений или время принятия решения.

Протокол двухфазной фиксации транзакций не гарантирует отсутствие блокировок в некоторых случаях. Блокировка, например, может иметь место в случае, когда на некотором узле истек тайм-аут после отправки своего согласия на фиксацию транзакции, но так и не получена информация о принятом глобальном решении от координатора, а те узлы, с которыми данный узел может обмениваться сообщениями, также не имеют информации о решении координатора. Подобные проблемы решаются в усовершенствованном варианте — трехфазном протоколе фиксации транзакций.

Протокол **трехфазной фиксации транзакций** не приводит к блокировкам в условиях отказа узлов, за исключением случая отказа всех узлов. Однако отказы линий связи могут привести к тому, что на разных узлах будут приняты разные решения, что является нарушением принципа неразрывности глобальной транзакции. Для использования протокола необходимо соблюдение трех условий: сеть является неразделяемой, всегда доступен по крайней мере один узел и отказы одновременно могут не более K узлов сети (такие системы называют K-устойчивыми).

Основным преимуществом протокола является устранение периода ожидания в состоянии неопределенности с момента подтверждения участниками своего согласия о фиксации транзакции до момента получения от координатора сообщения о фиксации или отмене транзакции. Для этого в протокол

фиксации транзакций введена дополнительная фаза — *предфиксации*. Таким образом протокол включает в себя три фазы: голосование, предфиксация транзакции и принятие глобального решения.

В целом взаимодействие координатора с участниками происходит следующим образом. После получения результатов голосования от всех участников координатор рассыпает сообщение о предфиксации PRE-COMMIT, извещающее о готовности всех участников зафиксировать результаты транзакции. В ответ на него каждый участник подтверждает получение этого сообщения. После приема подтверждений от всех участников координатор рассыпает команду глобальной фиксации транзакции. Обработка ситуации, когда некоторый участник потребовал отката транзакции, выполняется так же, как и в протоколе двухфазной фиксации транзакции.

Диаграммы переходов для координатора и участника приведены на рис. 4.9. На этапе предфиксации координатор и участники также переходят на некоторое время в состояние ожидания, однако все функционирующие процессы получают информацию о глобальном решении зафиксировать транзакцию еще до того, как первый процесс выполнит фиксацию результатов транзакции. Это позволяет участникам действовать независимо друг от друга в случае отказа.

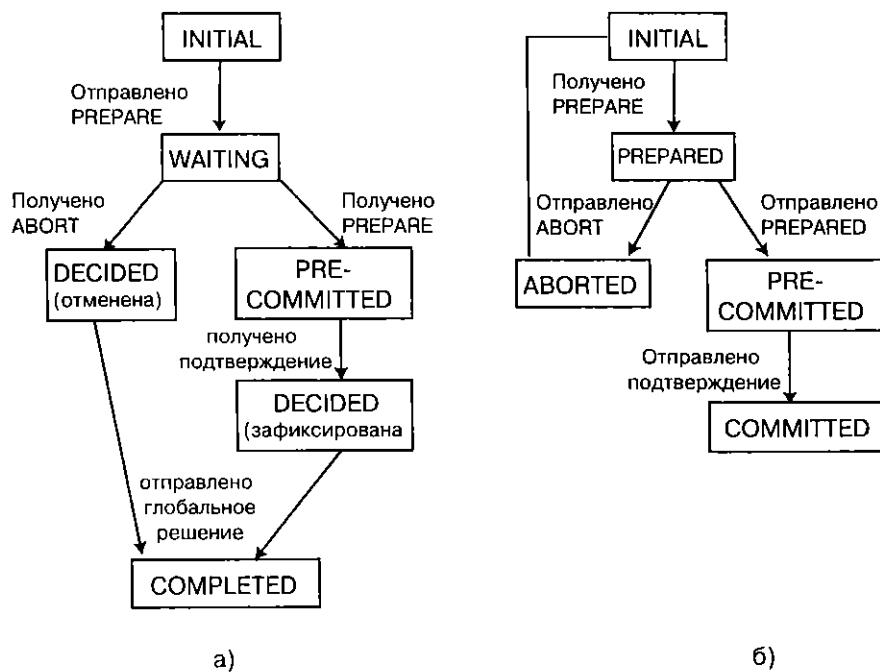


Рис. 4.9. Диаграммы переходов для протокола трехфазной фиксации транзакций:
а) координатор; б) участник

4.4. Информационные системы в локальных сетях

Локальная вычислительная сеть дает пользователю прежде всего возможность более эффективной организации групповых работ и совместного использования аппаратных ресурсов: принтеров, факсов, модемов, сканеров, дисков и т. д., а также программно-информационных ресурсов, в том числе баз данных. Рассмотрим основные *схемы организации работы с данными* в ЛВС.

Спектр возможных схем построения информационной системы в ЛВС существенно зависит от возможностей используемых СОС.

Основным сервисом современных ЛВС, независимо от типа управления в них (централизованные или одноранговые), является предоставление доступа одного компьютера (компьютера-клиента) к дискам, каталогам (папкам) и файлам другого компьютера (компьютера-сервера). Так, при обращении к внешнему файлу СОС сначала передает по сети файл (часть файла) в оперативную память компьютера, а по завершении работы с ним при необходимости возвращает обновленную версию. Кроме того, полезной функцией является возможность запуска на компьютере программ, хранящихся на другом компьютере.

Эти возможности примерно в равной мере предоставляются сетевыми ОС такими, как Novell NetWare 3.x (4.x), Windows NT и Windows 95/98. Более слабые сетевые пакеты и утилиты могут предоставлять доступ к информации без автоматического запуска программ. В этом случае пользователь должен сначала скопировать программы и файлы с другого компьютера на свой, после чего запустить программу.

Как и на отдельном компьютере, в ЛВС пользователь может управлять БД с помощью средств некоторой СУБД или работая с приложением. В общем случае приложение может выполняться под управлением СУБД или ее ядра, либо быть независимым и выполняться как автономная программа. Для удобства в дальнейшем под СУБД будем понимать любой из этих вариантов.

В локальной сети персональных ЭВМ выделяют следующие три варианта создания информационной системы:

- типа файл-сервер;
- типа клиент-сервер;
- основанные на технологии интранет;

Информационные системы типа *файл-сервер* можно строить двумя способами:

- с использованием *несетевых* СУБД, предназначенных для применения на отдельной машине;
- с использованием *сетевых* СУБД, разрабатываемых для применения в ЛВС.

Под *сетевой СУБД* здесь понимается система с произвольной моделью данных (не обязательно сетевой), ориентированная на использование в сети.

Программы *несетевой СУБД* и используемые ею данные могут храниться на компьютере-сервере (КС) и на компьютере-клиенте (КК).

Запуск и функционирование несетевой СУБД, хранящейся на КК и работающей с локальными данными, не отличается от обычного режима работы на отдельной ПЭВМ. Если используемые данные хранятся на КС, файловая система сетевой ОС «незаметно» для СУБД выполняет подгрузку нужного файла с удаленного компьютера. Заметим, что не каждая несетевая СУБД без проблем работает в среде любой сетевой ОС.

Если несетевая СУБД используется несколькими пользователями сети, то ее программы, а также БД или ее часть в целях экономии дисковой памяти эффективнее хранить на КС. Хранимую на КС БД будем называть *центральной БД* (ЦБД), а хранимую на КК БД — *локальной БД* (ЛБД). При запуске СУБД в таком варианте на каждый КК обычно пересыпается полная копия основной программы СУБД и один или несколько файлов ЦБД (рис. 4.10). После завершения работы файлы ЦБД должны пересыпаться с КК обратно на КС для согласования данных.

Существенным недостатком такого варианта применения несетевых СУБД является возможность нарушения целостности данных при одновременной работе с одной БД нескольких пользователей. Поскольку каждая копия СУБД функционирует «не зная» о работе других копий СУБД, тоника-

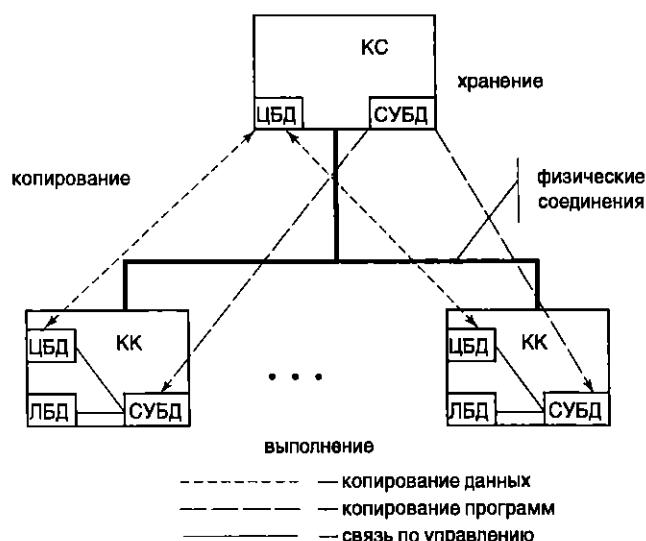


Рис. 4.10. Система типа файл-сервер с несетевой СУБД

ких мер по исключению возможных конфликтов не принимается. При этом элементарные операции чтения-записи одних и тех же файлов, как правило, контролирует сетевая ОС.

В качестве примеров несетевых СУБД можно назвать первые версии системы dBase III Plus, dBase IY и FoxBase.

Сетевые СУБД не имеют указанного недостатка, так как в них предусматривается «контроль соперничества» (concurrency control). Средства контроля позволяют осуществлять координацию доступа к данным, например, введением блокировок к файлам, записям и даже отдельным полям записей (рис. 4.11).

В сетевых СУБД с коллективным использованием файлов БД по-прежнему вся обработка информации производится на КК, а функции КС сводятся к предоставлению большой дисковой памяти. Такой подход нельзя считать эффективным, так как для обеспечения приемлемой скорости процесса обработки информации КК должен обладать высоким быстродействием и иметь большую емкость оперативной памяти. Кроме того, пересылка копий файлов БД и команд управления блокировками по линиям связи существенно увеличивает нагрузку на подсистему передачи данных, что снижает общую производительность сети.

Примерами сетевых СУБД являются: FoxPro 2.5 для Windows, dBase IY, Paradox 3.5 для DOS.

Информационные системы типа **клиент-сервер** отличаются от систем типа файл-сервер прежде всего тем, что программы СУБД функционально разделены на две части, называемые *сервером* и *клиентом*. Между клиентской и серверной частями системы возможны различные варианты распределения функций (подраздел 4.2).

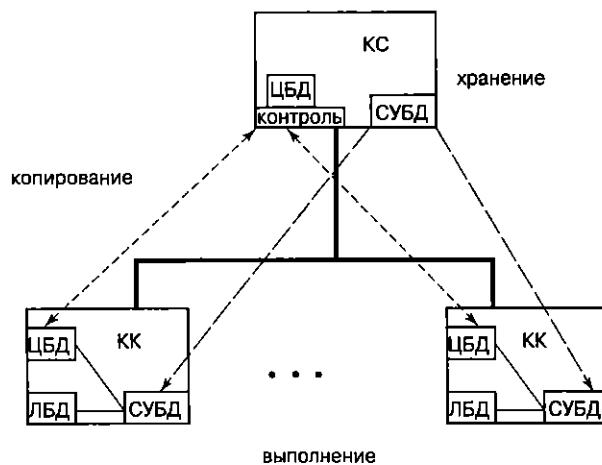


Рис. 4.11. Система типа файл-сервер с сетевой СУБД

Клиент, или *фронтальная программа*, отвечает за интерфейс с пользователем, для чего принимает и проверяет синтаксис вводимых пользователем запросов, преобразует их в команды запросов к серверной части, а при получении результатов выполняет обратное преобразование и отображение информации для пользователя.

В роли клиента выступает пользовательская (разрабатываемая для решения конкретной прикладной задачи программа) или готовая программа, имеющая интерфейс с серверной программой. В качестве готовых клиентских программ могут использоваться текстовые процессоры, табличные процессы и даже СУБД (например, Access, FoxPro и Paradox).

Сервер является основной программой, выполняющей функции управления и защиты данных в базе. На сервере принимаются запросы к базе со стороны клиентов с проверкой их полномочий, последующим выполнением и возвратом результатов, обеспечивается параллельный доступ к данным, контролируется соблюдение ограничений целостности, реализуются функции управления восстановлением. В случаях, когда вызов функций сервера выполняется на языке SQL, а именно так часто и происходит, его называют **SQL-сервером**.

В качестве сервера может использоваться ядро профессиональной реляционной СУБД (например, Informix 7.x и Sybase System 10) или некоторый SQL-сервер (например, Novell NetWare SQL и Microsoft SQL Server).

Структуру информационных систем типа клиент-сервер упрощенно можно представить как показано на рис. 4.12.

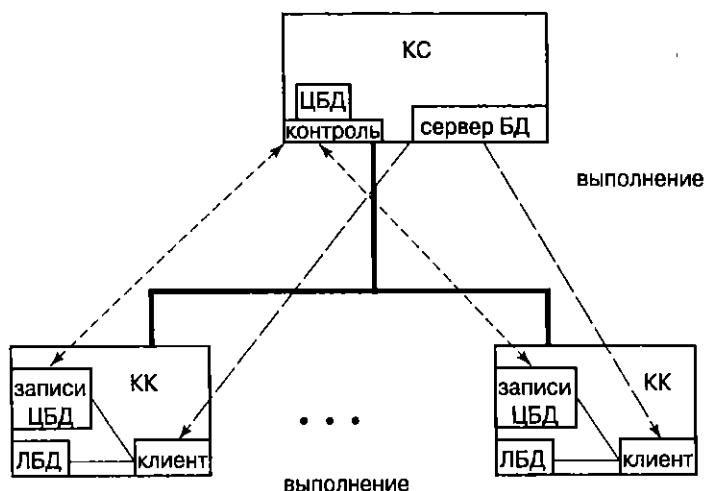


Рис. 4.12. Информационная система типа клиент-сервер

Основная часть обработки информации по формированию запросов, составлению отчетов, представлению данных в удобной для пользователя виде и т. д. выполняется на КК. Полные копии файлов БД из КС на КК и обратно (как в случае систем на базе сетевых СУБД) не пересылаются, поскольку для организации полноценного взаимодействия, как правило, достаточно иметь на КК необходимые в данный момент времени записи БД. Все это существенно снижает трафик в сети, ослабляет требования по ресурсам к КК, позволяет создавать более эффективные и надежные информационные системы.

В последнее время на компьютере-сервере, кроме собственно данных, хранят программы обработки данных и запросы. Это обеспечивает увеличение скорости обработки данных (программа обработки либо запрос находится «рядом» с данными), а также эффективность хранения и администрирования программ и запросов общего пользования в одном месте (на компьютере-сервере).

Хранимые на компьютере-сервере программы (процедуры) обработки данных называют **хранимыми процедурами**.

Разновидностью хранимой процедуры является так называемый **триггер**. Триггер (триггерная процедура) автоматически вызывается при возникновении определенных событий в БД. В качестве событий могут быть следующие: операции вставки, обновления и удаления отдельных записей, колонок и полей записей и другие. Примером триггера является программа, запускающая процесс посылки сообщения по электронной почте при достижении размера БД (количество записей) предельного значения.

В БД сервера некоторых систем можно хранить и сами запросы, называемые **хранимыми командами**. Совокупность хранимых команд — это поименованная совокупность команд, получаемых в результате компиляции SQL-запроса. Хранимые команды выполняются значительно быстрее, чем соответствующий SQL-запрос. Основная причина ускорения состоит в том, что при выполнении хранимых команд не требуетсяся синтаксический разбор запросов. Дополнительное ускорение выполнения запросов может быть получено в тех случаях, когда сервер БД не просто сохраняет коды команд запросов, а производит оптимизацию сохраняемого кода.

С хранимыми процедурами и командами связано понятие **курсора**, отличающееся от привычного понятия курсора как указателя текущей позиции на экране монитора. В разных СУБД — это близкие, но несколько отличающиеся понятия. Наиболее широко это понятие трактуется в СУБД SQLBase. Здесь курсор может означать следующее:

- идентификатор сеанса связи пользователя с СУБД;
- идентификатор хранимых команд и процедур;
- идентификатор результирующего множества;
- указатель текущей строки в результирующем множестве, обрабатываемом клиентским приложением.

Программы сервера (основные, хранимые процедуры и триггеры) могут быть выполнены как обычные программы (Windows 95/98), либо как специально загружаемые модули сетевой ОС (NLM-модули в сети Novell). Программы клиента в общем случае хранятся на КС или на КК, или в обоих местах.

В настоящее время среди программных продуктов существует огромное количество универсальных (в смысле пригодности работы с различными серверами БД) средств разработки систем типа клиент-сервер, к числу которых относятся: Delphi (Borland), Power Builder (Powersoft), ERwin (LogicWorks), Visual Basic (Microsoft), CA-Visual Objects (Computer Associates), SQL Windows (Gupta) и другие. Кроме того, существуют средства разработки в рамках определенных СУБД (например, для Oracle 7 – Designer/2000). Все подобные средства, как правило, относятся к CASE-системам (раздел 7).

При построении информационных систем типа клиент-сервер возникает проблема доступа со стороны СУБД или приложений, разработанных в одной среде, к данным, порожденным другой СУБД. В среде Windows эта проблема решается с помощью стандартного интерфейса ODBC (Open Database Connectivity – совместимость открытых баз данных) фирмы Microsoft. Основное его назначение заключается в обеспечении унифицированного доступа к локальным и удаленным базам данных различных производителей.

Схема доступа приложений к базам данных с помощью ODBC показана на рис. 4.13. Доступ приложения к данным происходит путем вызова на языке SQL стандартных функций интерфейса ODBC. На компьютере-клиенте при этом должна функционировать операционная система MS Windows с интерфейсом ODBC.

Взаимодействие приложения с данными производится с помощью менеджера (диспетчера) драйверов, который подключает необходимый драйвер в

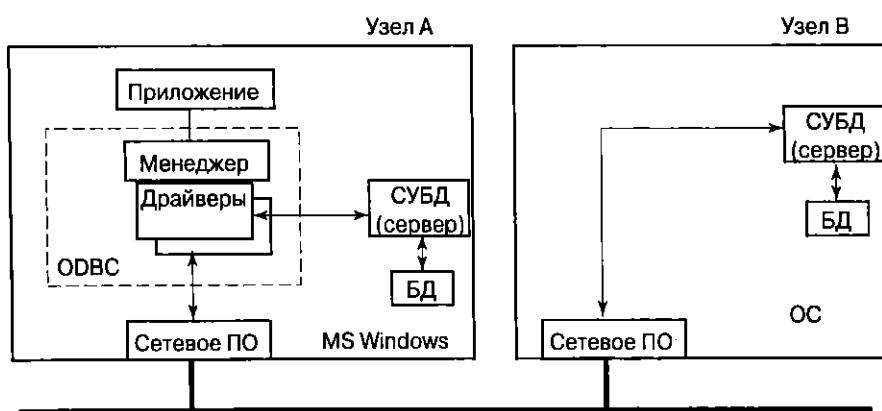


Рис. 4.13. Схема доступа к БД с помощью ODBC

соответствии с форматом данных СУБД. Драйвер СУБД, используя сетевые средства, как правило, коммуникационные модули конкретной СУБД, передает SQL-операторы серверу СУБД. Результаты выполнения запросов на сервере передаются обратно в приложение.

Рассмотренные схемы функционирования СУБД и внешних приложений касаются наиболее типичных вариантов их построения.

4.5. Информационные системы в Интернете и интранете

Обработка информации в среде Интернет существенно отличается от обработки информации в локальной сети и, тем более, на отдельном компьютере. Перечислим наиболее важные из них:

1. Большая протяженность коммуникационных линий, что сказывается на временных характеристиках обмена. Кроме того, большая удаленность лишает смысла загрузку программ с одного компьютера на другой и не позволяет выполнять пересылку больших объемов данных в реальном масштабе времени, как в сетевых СУБД локальных сетей.

2. Взаимодействие распределенных элементов ИС происходит с помощью обмена *пакетами* или *сообщениями*. Отдельные программные компоненты ИС могут быть одного или различных производителей. В последнем случае особую роль приобретает решение проблемы поддержки стандартов на сетевые протоколы и на язык SQL.

3. Сеть Интернет отличает от остальных глобальных сетей то, что по масштабам она больше всех других сетей (объединяет другие сети) и принципы ее организации оказывают существенное влияние на использование в сети баз данных.

Перед рассмотрением моделей и механизмов использования БД дадим краткую характеристику Интернета.

Характеристика Интернета

Основными видами услуг (сервиса), предоставляемых пользователям при подключении к Интернету, являются:

- электронная почта (E-mail);
- телеконференции (UseNet);
- система эмуляции удаленных терминалов (TelNet);
- поиск и передача двоичных файлов (FTP);
- поиск и передача текстовых файлов с помощью системы меню (Gopher);
- поиск и передача документов с помощью гипертекстовых ссылок (WWW или «Всемирная паутина»).

Создание и развитие этих способов связано с историей Интернета. Каждый из них характеризуется своими возможностями и различием в организации протоколов обмена информацией. Под протоколом, в общем случае, понимается набор инструкций, регламентирующих работу взаимосвязанных систем или объектов в сети.

Электронная почта (E-mail) – наиболее простой и доступный способ доступа в сети Интернет. Позволяет выполнять пересылку любых типов файлов (включая тексты, изображения, звуковые файлы) по адресам электронной почты в любую точку планеты за короткий промежуток времени в любое время суток. Для передачи сообщения необходимо знать электронный адрес получателя. Работа электронной почты основана на последовательной передаче информации по сети от одного почтового сервера к другому, пока сообщение не достигнет адресата. К достоинствам электронной почты относятся высокая оперативность и низкая стоимость. Недостаток электронной почты состоит в ограниченности объема пересылаемых файлов.

Система телеконференций UseNet разработана как система обмена текстовой информацией. Она позволяет всем пользователям Интернета участвовать в групповых дискуссиях, называемых телеконференциями, в которых обсуждаются всевозможные проблемы. Сейчас в мире насчитывается более 10 тысяч телеконференций. Информация, посылаемая в телеконференции, становится доступной любому пользователю Интернета, обратившемуся в данную телеконференцию. В настоящее время телеконференции позволяют передавать файлы любых типов. Для работы с телеконференциями наиболее часто используются средства программ просмотра и редактирования Web-документов.

TelNet – это протокол, позволяющий одному компьютеру использовать ресурсы другого (удаленного) компьютера. Другими словами – это протокол удаленного терминального доступа в сети.

FTP (File Transfer Protocol) – это протокол, позволяющий передавать файлы произвольного формата между двумя компьютерами сети. Программное обеспечение FTP разработано по архитектуре «клиент-сервер» и разделено на две части: серверную (FTP-сервер) и клиентскую. FTP-клиент, в общем случае, позволяет пользователям просматривать файловую систему FTP-сервера и производить с ней обмен файлами (выгружать файлы своего компьютера, загружать, переименовывать и удалять файлы удаленного компьютера). Достоинством данного протокола является возможность передачи файлов любого типа, в том числе исполняемых программ. К недостатку протокола FTP следует отнести необходимость априорного знания местоположения отыскиваемой информации (FTP-адреса).

Протокол **Gopher** и реализующее его программное обеспечение предоставляют пользователям возможность работы с информационными ресурсами, не зная заранее их местонахождение. Для начала работы по этому протоколу до-

статочно знать адрес одного Gopher-сервера. В дальнейшем работа заключается в выборе команд, представленных в виде простых и понятных меню. При этом пункты меню одного сервера могут содержать ссылки на меню других серверов, что облегчает поиск требуемой информации в сети Интернет. Во время работы с системой Gopher программа-клиент не поддерживает постоянного соединения с Gopher-сервером, что позволяет экономить сетевые ресурсы.

WWW (World Wide Web – всемирная паутина) представляет собой самое популярное и современное средство организации сетевых ресурсов. Она строится на основе гипертекстового представления информации.

Гипертекстовый документ (гипертекст) представляет собой текст, содержащий ссылки на другие фрагменты текстов произвольных документов, в том числе и этого документа. Гипертекстовый документ подготавливается на стандартизированном языке HTML (HyperText Markup Language – язык разметки гипертекста). Он состоит из страниц (web-страниц), доступ к которым основан на протоколе передачи гипертекста (HyperText Transfer Protocol, HTTP).

Простейшим примером гипертекста является книга, оглавление которой содержит ссылки (внутренние) в виде номеров страниц на разделы, подразделы, пункты книги, кроме того, в книге имеются внешние ссылки на другие используемые источники информации.

Фрагмент документа может включать в себя информацию в виде обычного текста, графического изображения, звука и движущегося изображения (анимации). Гипертекст с нетекстовыми документами часто называют **ги-пермедиа**.

Важнейшим свойством гипертекста является наличие в нем ссылок на документы, размещаемые на территориально удаленных компьютерах. Документы могут создаваться и редактироваться различными людьми. Вся совокупность взаимосвязанных документов образует гигантскую «паутину». Эта модель подобна модели окружающего нас бесконечного информационного пространства, когда нет строгой иерархии связей, а есть множество связей без начала и конца.

Работа сети Интернет основана на использовании протокола TCP/IP (Transmission Control Protocol/Internet Protocol – Протокол управления передачей данных/Протокол Интернет), который используется для передачи данных в глобальной сети и во многих локальных сетях. TCP/IP в основном реализует функции транспортного и сетевого уровней модели OSI (подраздел 4.1). Он представляет собой семейство коммуникационных протоколов, которые по назначению можно разделить на следующие группы:

- транспортные протоколы, служащие для управления передачей данных между двумя компьютерами;
- протоколы маршрутизации, обрабатывающие адресацию данных и определяющие кратчайшие доступные пути к адресату;

- протоколы поддержки сетевого адреса, предназначенные для идентификации компьютера по его уникальному номеру или имени;
- прикладные протоколы, обеспечивающие получение доступа к всевозможным сетевым услугам;
- шлюзовые протоколы, помогающие передавать по сети сообщения о маршрутизации и информацию о состоянии сети, а также обрабатывать данные для локальных сетей;
- другие протоколы, не относящиеся к указанным категориям, но обеспечивающие клиенту удобство работы в сети.

Доступ пользователей к ресурсам Интернета обычно производится с помощью программ-навигаторов, или **обозревателей** (browser). В настоящее время к числу наиболее популярных программ этого класса относятся следующие: Netscape Navigator/Communicator (Netscape) и MS Explorer (Microsoft). Хотя эти программы основаны на использовании протокола HTTP, они предоставляют простой доступ к другим сервисам Интернета: электронной почте, новостям и т. д.

Обозреватель, обеспечивая доступ пользователя к ресурсам сети, по существу является программой-клиентом (или Web-клиентом). Программой, предоставляющей информационные ресурсы, является Web-сервер. Именно он осуществляет основную работу по сбору и получению информации из разных источников, после чего в стандартном виде предоставляет ее Web-клиенту. Рассмотрим организацию выбора информации для пользователя, если она находится в базах данных.

Базы данных в Интернете и интранете

Технология интранет по существу представляет собой технологию Интернет, перенесенную в среду корпоративных ИС. Архитектура информационных систем в Интернете и интранете является результатом эволюционного перехода от первых много пользовательских централизованных вычислительных систем (мэйнфреймов) через системы типа клиент-сервер к распределенным системам с централизованной обработкой и подготовкой информации к непосредственному потреблению. Рассмотрим кратко указанные этапы эволюции.

1. В **мэйнфреймах** (рис. 4.14) вычислительные ресурсы, хранимые данные и программы обработки информации сконцентрированы в одной ЭВМ. Основным средством доступа был алфавитно-цифровой терминал (дисплей), управляемый ЭВМ. Вся обработка информации и подготовка ее к выдаче выполнялись на центральной ЭВМ. С терминалов, как правило, в машину передавались коды нажатия клавиш или содержимое буфера экрана, а обратно на терминал — пересылались отображаемые экраны с соответствующими кодами управления отображением.

Достоинством системы является простота администрирования, защиты информации и модификации системы, к недостаткам можно отнести высокую загрузку процессоров и линий связи (как следствие — невысокую реакцию си-



Рис. 4.14. Централизованная многопользовательская система

стемы при большом количестве пользователей), низкую надежность (выход из строя ЭВМ приводит к полному отказу всей системы), сложность масштабирования системы и некоторые другие.

2. Исторически следующим решением в области информационных систем была архитектура *клиент-сервер* (рис. 4.15).

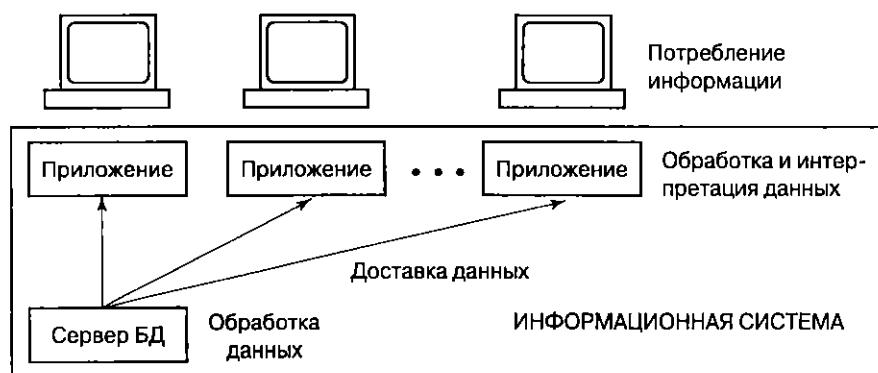


Рис. 4.15. Системы типа клиент-сервер

В этих системах место терминала заняла ПЭВМ, а мэйнфрейма — компьютер-сервер. Ранее мы рассматривали спектр моделей подобных систем с различным распределением функций между компонентами. Если не брать во внимание модель «распределенного представления» (по сути повторяет модель централизованной многопользовательской системы), то можно заключить, что системы типа клиент-сервер имеют следующие *достоинства*: высокая живучесть и надежность, легкость масштабирования, качественный пользовательский интерфейс, возможность одновременной работы с несколькими приложениями, высокие характеристики оперативности обработки информации.

Основным *недостатком* клиент-серверных систем является то, что они ориентированы на данные, а не на информацию. Это требует от пользователя

знания не только предметной области, а и специфики используемой прикладной программы.

Существенным недостатком можно считать также сложность переноса таких систем на другие компьютерные платформы и интеграцию с другими пакетами из-за «закрытости» используемых протоколов взаимодействия компонентов систем.

Еще один недостаток заключается в сложности администрирования системы и ее уязвимости при непредсказуемых или злонамеренных действиях пользователя или компьютерных вирусов.

3. **Корпоративные системы инTRANET**, в отличие от систем клиент-сервер, ориентированы не на данные, а на информацию в ее окончательном и пригодном для использования неквалифицированным пользователем виде (рис. 4.16).



Рис. 4.16. Системы, поставляющие информацию

Новые системы объединяют в себе преимущества централизованных многопользовательских систем и систем типа клиент-сервер. Им присущи следующие черты:

- на сервере порождается информация, пригодная для использования, а не данные (например, в случае СУБД – записи БД);
- при обмене между клиентской и серверной частями используется протокол открытого стандарта, а не какой-то конкретной фирмы;
- прикладная система находится на сервере, и поэтому для работы пользователя на компьютере-клиенте достаточно иметь программу-навигатор (могут быть и другие решения, когда часть обработки производится на компьютере-клиенте).

В случае, когда источником информации в Интернете и интранете являются БД, имеет место взаимодействие компонентов WWW и традиционных СУБД. Типовыми простейшими схемами организации функционирования программных компонентов, использующих данные из некоторой базы, в настоящее время можно считать следующие три: на стороне Web-клиента (рис. 4.17 а), на стороне Web-сервера (рис. 4.17 б) и на стороне сервера приложений (рис. 4.17 в).

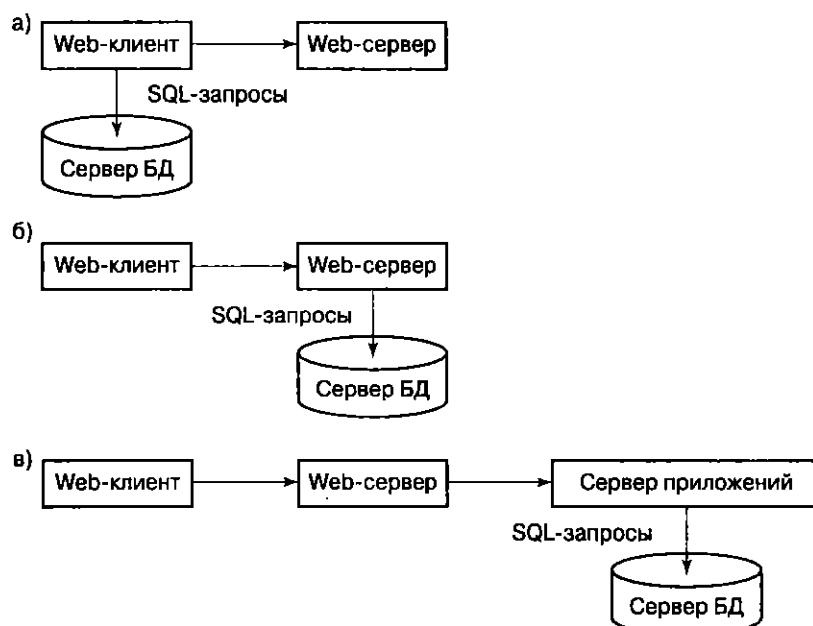


Рис. 4.17. Модели доступа к базе данных в Интернете

При доступе к БД **на стороне клиента** основным средством реализации механизмов взаимодействия Web-клиента и сервера БД является язык Java. Кроме того, могут использоваться элементы управления ActiveX. В качестве вспомогательных средств обработки информации на клиентской стороне (но не для взаимодействия с базами данных) часто используются языки сценариев JavaScript, JScript и VBScript, разработанные для расширения возможностей декларативного языка HTML (в HTML нет операторов присваивания, сравнения, математических функций и пр.) на основе добавления процедурных средств. Программы-сценарии выполняются на компьютере Web-браузером в режиме интерпретации.

Для обращений к серверам БД из Java-программ разработан стандарт JDBC (Java DataBase Connectivity – совместимость баз данных для Java), основанный

ный на концепции ODBC. Стандарт JDBC разработан фирмами Sun/JavaSoft и обеспечивает универсальный доступ к различным базам данных на языке Java.

В модели доступа к БД **на стороне сервера** обращение к серверу БД обычно производится путем вызова программами Web-сервера внешних по отношению к ним программ в соответствии с соглашениями одного из интерфейсов: CGI (Common Gateway Interface – общий шлюзовый интерфейс), FastCGI или API (Application Program Interface – интерфейс прикладного программирования).

Внешние программы взаимодействуют каким-либо образом с сервером БД на языке SQL, например, непосредственно обращаясь к конкретному серверу или используя драйвер ODBC (см. подраздел 9.3).

Внешние программы пишутся на обычных языках программирования типа C, C++ и Паскаль или специализированных языках типа Perl или PHP. Программы, разработанные в соответствии с интерфейсом CGI, называются **CGI-схемариями**.

Кроме того, для организации доступа серверных программ к информации из БД могут использоваться технологии динамического построения Web-страниц (ASP, PHP и IDC/HTX-страницы) на основе информации БД.

Доступе к БД **на стороне сервера приложений** обычно применяется при использовании серверов приложений. Основным языком разработки распределенных приложений в этом случае можно считать язык Java, а также технологии CORBA и Enterprise JavaBeans.

Из трех рассмотренных схем однозначного предпочтения тому или иному варианту отдать нельзя. Все зависит от целей и условий разработки клиент-серверных программ (наиболее существенными оказываются аппаратно-программная платформа, вида Web-сервера, нагрузка на Web-сервер, а также характер решаемых задач).

Недостатком первой модели является то, что клиентская часть системы оказывается более нагруженной, чем во второй модели. Кроме того, в некоторых случаях (например, при использовании технологии ActiveX) повышается угроза нарушению защиты информации на клиентской стороне. В то же время разгружается Web-сервер, что является **достоинством**.

Достоинством модели доступа на стороне сервера является сравнительная простота клиентских программ и удобство администрирования системы, так как основная часть программного обеспечения находится на машине Web-сервера. Очевидным **недостатком** системы является возможное ухудшение характеристик оперативности получения информации при большой нагрузке на Web-сервер и нехватке его мощности.

В третьей схеме предпринята попытка преодолеть недостатки второй схемы в том случае, когда планируется большая нагрузка на Web-сервер.

Подробно методы и средства организации работы с базами данных в Интернете рассмотрены в части 4.

Контрольные вопросы и задания

1. Назовите типичные варианты разделения функций между компьютером-сервером и компьютером-клиентом для двухзвенной модели.
2. Охарактеризуйте модель удаленного доступа к данным.
3. Назовите достоинства и недостатки модели сервера баз данных.
4. Изобразите структурную схему трехзвенной модели сервера приложений.
5. Каково назначение мониторов транзакций?
6. Изобразите схему и охарактеризуйте модель монитора транзакций.
7. Назовите основные технологии децентрализованного управления БД.
8. Опишите динамику функционирования модели распределенной БД.
9. Укажите достоинства и недостатки модели распределенной БД.
10. Опишите протокол двухфазной фиксации транзакций.
11. Опишите модель тиражирования данных.
12. Опишите протокол трехфазной фиксации транзакций.
13. Назовите основные методы доступа к данным и укажите случаи предпочтительного их использования.
14. Приведите пример типичного набора блокировок объектов БД.
15. Укажите правила совмещения блокировок.
16. Назовите основные разновидности тупиков.
17. Приведите пример взаимного тупика в распределенной БД.
18. Укажите основные варианты создания информационной системы в локальной сети.
19. Опишите схему функционирования информационной системы типа файл-сервер с несетевой СУБД.
20. Как организуется обработка в информационных системах типа файл-сервер с сетевой СУБД?
21. Опишите схему функционирования информационной системы типа клиент-сервер.
22. Каково назначение хранимых процедур и триггеров?
23. Дайте понятия хранимых команд и курсора.
24. Как организуется доступ к данным с помощью интерфейса ODBC?
25. Дайте общую характеристику сети Internet.
26. Укажите основные модели доступа к БД в сети Internet.
27. Какие языки программирования используются для доступа к БД в Internet?
28. Каково назначение CGI-сценариев?
29. Охарактеризуйте технологию intranet.

Литература

1. Вейд А. Стандарты объектных запросов // Системы Управления Базами Данных, № 4, 1996. – С. 89-97.
2. Дейт К.Дж. Введение в системы баз данных / Пер. с англ. – 6-е изд. – К.: Диалектика, 1998. – 784 с.
3. Елисеев В, Ладыженский Г. Введение в Интранет // Системы Управления Базами Данных, № 5-6, 1996. – С. 19-43.
4. Калиниченко Л. Стандарт систем управления объектными базами данных ODMG: краткий обзор и оценка состояния // Системы Управления Базами Данных, № 1, 1996. – С. 102-109.
5. Конполи Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Пер. с англ. – 3-е издание. – М.: Издательский дом «Вильямс», 2003. – 1440 с.
6. Кузнецов С. Доступ к базам данных с использованием технологии WWW // Системы Управления Базами Данных, № 5-6, 1996. – С. 4-9.
7. Ладыженский Г. Системы управления базами данных – коротко о главном // Системы Управления Базами Данных, № 4, 1995. – С. 123-141.
8. Олейников А.Я. Открытые системы: концепция и реальность // Открытые системы, № 4, 1993. – С. 53-58.
9. Орлик С.В. Borland Delphi как средство разработки масштабируемых приложений // Системы Управления Базами Данных, № 4, 1995. – С. 50-56.
10. Основы современных компьютерных технологий: Учебник / Под ред. проф. Хомоненко А. Д. Авторы: Брякалов Г. А., Войцеховский С. В., Воробьев Е. Г., Гофман В. Э., Гридин В. В., Дрюков Ю. П., Замула А. А., Захаров А. И., Компаниец Р. И., Липецких А. Г., Рыжиков Ю. И., Хомоненко А. Д., Цыганков В. М. – СПб: КОРОНА прнт, 2005. – 672 с.
11. Роберт Сигнор, Михаэль О. Стегман. Использование ODBC для доступа к базам данных / Пер. с англ. – М.: БИНОМ; НАУЧНАЯ КНИГА. – 384 с.
12. Системы управления базами данных и знаний: Справ. изд. / Наумов А.Н., Вендрров А.М., Иванов В.К. и др.; Под ред. Наумова А.Н. – М.: Финансы и статистика, 1991. – 352 с.
13. Советов Б. Я., Цехановский В. В., Чертовский В. Д. Базы данных. Теория и практика. – М.: Высшая школа, 2005.

5. Проектирование баз данных

В настоящем разделе рассматриваются вопросы проектирования реляционных баз данных. Даётся характеристика проблем проектирования и подходов к их решению.

5.1. Проблемы проектирования

Проектирование информационных систем, включающих в себя базы данных, осуществляется на физическом и логическом уровнях. Решение проблем проектирования на *физическом уровне* во многом зависит от используемой СУБД, зачастую автоматизировано и скрыто от пользователя. В ряде случаев пользователю предоставляется возможность настройки отдельных параметров системы, которая не составляет большой проблемы.

Логическое проектирование заключается в определении числа и структуры таблиц, формировании запросов к БД, определении типов отчетных документов, разработке алгоритмов обработки информации, создании форм для ввода и редактирования данных в базе и решении ряда других задач.

Решение задач логического проектирования БД в основном определяется спецификой задач предметной области. Наиболее важной здесь является проблема структуризации данных, на ней мы сосредоточим основное внимание.

При проектировании *структур данных* для автоматизированных систем можно выделить три основных подхода:

1. Сбор информации об объектах решаемой задачи в рамках одной таблицы (одного отношения) и последующая декомпозиция ее на несколько взаимосвязанных таблиц на основе процедуры нормализации отношений.

2. Формулирование знаний о системе (определение типов исходных данных и их взаимосвязей) и требований к обработке данных, получение с помощью CASE-системы (системы автоматизации проектирования и разработки баз данных) готовой схемы БД или даже готовой прикладной информационной системы.

3. Структурирование информации для использования в информационной системе в процессе проведения системного анализа на основе совокупности правил и рекомендаций.

Ниже мы рассмотрим первый из названных подходов, являющийся классическим и исторически первым. Прежде всего охарактеризуем основные проблемы, имеющие место при определении структур данных в отношениях реляционной модели.

Избыточное дублирование данных и аномалии

Следует различать простое (неизбыточное) и избыточное дублирование данных. Наличие первого из них допускается в базах данных, а избыточное дублирование данных может приводить к проблемам при обработке данных. Приведем примеры обоих вариантов дублирования.

Пример *неизбыточного дублирования* данных представляет приведенное на рис. 5.1 отношение C_T с атрибутами Сотрудник и Телефон. Для сотрудников, находящихся в одном помещении, номера телефонов совпадают. Номер телефона 4328 встречается несколько раз, хотя для каждого служащего номер телефона уникален. Поэтому ни один из номеров не является избыточным. Действительно, при удалении одного из номеров телефонов будет потеряна информация о том, по какому номеру можно дозвониться до одного из служащих.

C_T	
Сотрудник	Телефон
Иванов И.М.	3721
Петров М.И.	4328
Сидоров Н.Г.	4328
Егоров В.В.	4328

Рис. 5.1. Неизбыточное дублирование

Пример *избыточного дублирования (избыточности)* представляет приведенное на рис. 5.2а отношение C_T_H, которое, в отличие от отношения C_T, дополнено атрибутом H_комн (номер комнаты сотрудника). Естественно предположить, что все служащие в одной комнате имеют один и тот же телефон. Следовательно, в рассматриваемом отношении имеется избыточное дублирование данных. Так, в связи с тем, что Сидоров и Егоров находятся в той же комнате, что и Петров, их номера можно узнать из кортежа со сведениями о Петрове.

На рис. 5.2б приведен пример неудачного отношения C_T_H, в котором вместо телефонов Сидорова и Егорова поставлены прочерки (неопределенные значения). Неудачность подобного способа исключения избыточности заключается в следующем. Во-первых, при программировании придется потратить дополнительные усилия на создание механизма поиска информации

С_Т_Н		
Сотрудник	Телефон	Н_комн
Иванов И.М.	3721	109
Петров М.И.	4328	111
Сидоров Н.Г.	4328	111
Егоров В.В.	4328	111

С_Т_Н		
Сотрудник	Телефон	Н_комн
Иванов И.М.	3721	109
Петров М.И.	4328	111
Сидоров Н.Г.	—	111
Егоров В.В.	—	111

Рис. 5.2. Избыточное дублирование

для прочерков таблицы. Во-вторых, память все равно выделяется под атрибуты с прочерками, хотя дублирование данных и исключено. В-третьих, что особенно важно, при исключении из коллектива Петрова кортеж со сведениями о нем будет исключен из отношения, а значит, уничтожена информация о телефоне 111-й комнаты, что недопустимо.

Возможный способ выхода из данной ситуации приведен на рис. 5.3. Здесь показаны два отношения С_Н и Н_Т, полученные путем декомпозиции исходного отношения С_Т_Н. Первое из них содержит информацию о номерах комнатах, в которых располагаются сотрудники, а второе – информацию о номерах телефонов в каждой из комнат. Теперь, если Петрова и уволят из учреждения и, как следствие этого, удалят всякую информацию о нем из баз данных учреждения, это не приведет к утере информации о номере телефона в 111-й комнате.

Т_Н		С_Н	
Телефон	Н_комн	Сотрудник	Н_комн
3721	109	Иванов И.М.	109
4328	111	Петров М.И.	111

Рис. 5.3. Исключение избыточного дублирования

Процедура декомпозиции отношения С_Т_Н на два отношения С_Н и Н_Т является основной процедурой нормализации отношений.

Избыточное дублирование данных создает проблемы при обработке кортежей отношения, называемые Э. Коддом «аномалиями обновления отношения». Он показал, что для некоторых отношений проблемы возникают при попытке удаления, добавления или редактирования их кортежей.

Аномалиями будем называть такую ситуацию в таблицах БД, которая приводит к противоречиям в БД либо существенно усложняет обработку данных.

Выделяют три основные вида аномалий: аномалии модификаций (или редактирования), аномалии удаления и аномалии добавления.

Аномалии модификации проявляются в том, что изменение значения одного данного может повлечь за собой просмотр всей таблицы и соответствующее изменение некоторых других записей таблицы.

Так, например, изменение номера телефона в комнате 111 (рис. 5.2а), что представляет собой один единственный факт, потребует просмотра всей таблицы С_Т_Н и изменения поля Н_комн согласно текущему содержимому таблицы в записях, относящихся к Петрову, Сидорову и Егорову.

Аномалии удаления состоят в том, что при удалении какого-либо данного из таблицы может пропасть и другая информация, которая не связана напрямую с удаляемым данным.

В той же таблице С_Т_Н удаление записи о сотруднике Иванове (например, по причине увольнения или ухода на заслуженный отдых) приводит к исчезновению информации о номере телефона, установленного в 109-й комнате.

Аномалии добавления возникают в случаях, когда информацию в таблицу нельзя поместить до тех пор, пока она неполная, либо вставка новой записи требует дополнительного просмотра таблицы.

Примером может служить операция добавления нового сотрудника все в ту же таблицу С_Т_Н. Очевидно, будет противовесственным хранение сведений в этой таблице только о комнате и номере телефона в ней, пока никто из сотрудников не помещен в нее. Более того, если в таблице С_Т_Н поле Служащий является ключевым, то хранение в ней неполных записей с отсутствующей фамилией служащего просто недопустимо из-за неопределенности значения ключевого поля.

Вторым примером возникновения аномалии добавления может быть ситуация включения в таблицу нового сотрудника. При добавлении таких записей для исключения противоречий желательно проверить номер телефона и соответствующий номер комнаты хотя бы с одним из сотрудников, сидящих с новым сотрудником в той же комнате. Если же окажется, что у нескольких сотрудников, сидящих в одной комнате, имеются разные телефоны, то вообще не ясно, что делать (то ли в комнате несколько телефонов, то ли какой-то из номеров ошибочный).

Формирование исходного отношения

Проектирование БД начинается с определения всех объектов, сведения о которых будут включены в базу, и определения их атрибутов. Затем атрибуты сводятся в одну таблицу – исходное отношение.

Пример. Формирование исходного отношения.

Предположим, что для учебной части факультета создается БД о преподавателях. На первом этапе проектирования БД в результате общения с за-

казчиком (заведующим учебной частью) должны быть определены содержащиеся в базе сведения о том, как она должна использоваться и какую информацию заказчик хочет получать в процессе ее эксплуатации. В результате устанавливаются атрибуты, которые должны содержаться в отношениях БД, и связи между ними. Перечислим имена выделенных атрибутов и их краткие характеристики:

ФИО – фамилия и инициалы преподавателя. Исключаем возможность совладения фамилии и инициалов у преподавателей.

Должн – должность, занимаемая преподавателем.

Оклад – оклад преподавателя.

Стаж – преподавательский стаж.

Д_Стаж – надбавка за стаж.

Каф – номер кафедры, на которой числится преподаватель.

Предм – название предмета (дисциплины), читаемого преподавателем.

Группа – номер группы, в которой преподаватель проводит занятия.

ВидЗан – вид занятий, проводимых преподавателем в учебной группе.

Одно из требований к отношениям заключается в том, чтобы все атрибуты отношения имели атомарные (простые) значения. В исходном отношении каждый атрибут кортежа также должен быть простым. Пример исходного отношения ПРЕПОДАВАТЕЛЬ приведен на рис. 5.4.

ПРЕПОДАВАТЕЛЬ

ФИО	Должн	Оклад	Стаж	Д_Стаж	Каф	Предм	Группа	ВидЗан
Иванов И.М.	преп	500	5	100	25	СУБД	256	Практ
Иванов И.М.	преп	500	5	100	25	ПЛ/1	123	Практ
Петров М.И.	ст.преп	800	7	100	25	СУБД	256	Лекция
Петров М.И.	ст.преп	800	7	100	25	Паскаль	256	Практ
Сидоров Н.Г.	преп	500	10	150	25	ПЛ/1	123	Лекция
Сидоров Н.Г.	преп	500	10	150	25	Паскаль	256	Лекция
Егоров В.В.	преп	500	5	100	24	ПЭВМ	244	Лекция

Рис. 5.4. Исходное отношение ПРЕПОДАВАТЕЛЬ

Указанное отношение имеет следующую схему ПРЕПОДАВАТЕЛЬ (ФИО, Должн, Оклад, Стаж, Д_Стаж, Каф, Предм, Группа, ВидЗан).

Исходное отношение ПРЕПОДАВАТЕЛЬ содержит избыточное дублирование данных, которое и является причиной аномалий редактирования. Различают избыточность явную и неявную.

Явная избыточность заключается в том, что в отношении ПРЕПОДАВАТЕЛЬ строки с данными о преподавателях, проводящих занятия в нескольких группах, повторяются соответствующее число раз. Например, в отношении ПРЕПОДАВАТЕЛЬ все данные по Иванову повторяются дважды. Поэтому, если Иванов И.М. станет старшим преподавателем, то этот факт должен быть отражен в обеих строках. В противном случае будет иметь место противоречие в данных, что является примером аномалии редактирования, обусловленной явной избыточностью данных в отношении.

Неявная избыточность в отношении ПРЕПОДАВАТЕЛЬ проявляется в одинаковых окладах у всех преподавателей и в одинаковых добавках к окладу за одинаковый стаж. Поэтому, если при изменении окладов за должность с 500 на 510 это значение изменят у всех преподавателей, кроме, например, Сидорова, то база станет противоречивой. Это пример аномалии редактирования для варианта с неявной избыточностью.

Средством исключения избыточности в отношениях и, как следствие, аномалий является нормализация отношений, рассмотрим ее более подробно.

5.2. Метод нормальных форм

Проектирование БД является одним из этапов жизненного цикла информационной системы. Основной задачей, решаемой в процессе проектирования БД, является задача нормализации ее отношений. Рассматриваемый ниже метод нормальных форм является классическим методом проектирования реляционных БД. Этот метод основан на фундаментальном в теории реляционных баз данных понятии зависимости между атрибутами отношений.

Зависимости между атрибутами

Рассмотрим основные виды зависимостей между атрибутами отношений: функциональные, транзитивные и многозначные.

Понятие функциональной зависимости является базовым, так как на его основе формулируются определения всех остальных видов зависимостей.

Атрибут В **функционально зависит** от атрибута А, если каждому значению А соответствует в точности одно значение В. Математически функциональная зависимость В от А обозначается записью $A \rightarrow B$. Это означает, что во всех кортежах с одинаковым значением атрибута А атрибут В будет иметь также одно и то же значение. Отметим, что А и В могут быть составными – состоять из двух и более атрибутов.

В отношении на рис. 5.4 можно выделить функциональные зависимости между атрибутами $\text{ФИО} \rightarrow \text{Каф}$, $\text{ФИО} \rightarrow \text{Должи}$, $\text{Должн} \rightarrow \text{Оклад}$ и другие. Наличие функциональной зависимости в отношении определяется природой вещей, информация о которых представлена кортежами отношения. В отно-

шении на рис. 5.4 ключ является составным и состоит из атрибутов ФИО, Предмет, Группа.

Функциональная взаимозависимость. Если существует функциональная зависимость вида $A \rightarrow B$ и $B \rightarrow A$, то между А и В имеется взаимно однозначное соответствие, или функциональная взаимозависимость. Наличие функциональной взаимозависимости между атрибутами А и В обозначим как $A \leftrightarrow B$ или $B \leftrightarrow A$.

Пример. Пусть имеется некоторое отношение, включающее два атрибута, функционально зависящие друг от друга. Это серия и номер паспорта (N) и фамилия, имя и отчество владельца (ФИО). Наличие функциональной зависимости поля ФИО от N означает не только тот факт, что значение поля N однозначно определяет значение поля ФИО, но и то, что одному и тому же значению поля N соответствует только единственное значение поля ФИО. Понятно, что в данном случае действует и обратная ФЗ: каждому значению поля ФИО соответствует только одно значение поля N. В данном примере предполагается, что ситуация наличия полного совпадения фамилий, имен и отчеств двух людей исключена.

Если отношение находится в 1НФ, то все неключевые атрибуты функционально зависят от ключа с различной степенью зависимости.

Частичной зависимостью (частичной функциональной зависимости) называется зависимость неключевого атрибута от части составного ключа. В рассматриваемом отношении атрибут Должн находится в функциональной зависимости от атрибута ФИО, являющегося частью ключа. Тем самым атрибут Должн находится в частичной зависимости от ключа отношения.

Альтернативным вариантом является **полная функциональная зависимость** неключевого атрибута от всего составного ключа. В нашем примере атрибут ВидЗан находится в полной функциональной зависимости от составного ключа.

Атрибут С зависит от атрибута А **транзитивно** (существует **транзитивная зависимость**), если для атрибутов А, В, С выполняются условия $A \rightarrow B$ и $B \rightarrow C$, но обратная зависимость отсутствует. В отношении на рис. 5.4 транзитивной зависимостью связаны атрибуты:

ФИО → Должн → Оклад

Между атрибутами может иметь место многозначная зависимость.

В отношении R атрибут В **многозначно зависит** от атрибута А, если каждому значению А соответствует множество значений В, не связанных с другими атрибутами из R.

Многозначные зависимости могут быть «один ко многим» (1:M), «многие к одному» (M:1) или «многие ко многим» (M:M), обозначаемые соответственно: $A \Rightarrow B$, $A \Leftarrow B$ и $A \leftrightarrow B$.

Например, пусть преподаватель ведет несколько предметов, а каждый предмет может вестись несколькими преподавателями, тогда имеет место зависимость ФИО \leftrightarrow Предмет. Так, из таблицы 7.2, приведенной на рис. 5.4., видно, что преподаватель Иванов И.М. ведет занятия по двум предметам, а дисциплина СУБД – читается двумя преподавателями: Ивановым И.М. и Петровым М.И.

Замечание. В общем случае между двумя атрибутами одного отношения могут существовать зависимости: 1:1, 1:M, M:1 и M:M. Поскольку зависимость между атрибутами является причиной аномалий, стараются расчленить отношения с зависимостями атрибутов на несколько отношений. В результате образуется совокупность связанных отношений (таблиц) со связями вида 1:1, 1:M, M:1 и M:M (подраздел 3.3). Связи между таблицами отражают зависимости между атрибутами различных отношений.

Взаимно независимые атрибуты. Два или более атрибута называются взаимно независимыми, если ни один из этих атрибутов не является функционально зависимым от других атрибутов.

В случае двух атрибутов отсутствие зависимости атрибута A от атрибута B можно обозначить так: A $\rightarrow\rightarrow$ B. Случай, когда A $\rightarrow\rightarrow$ B и B $\rightarrow\rightarrow$ A, можно обозначить A \leftrightarrow B.

Выявление зависимостей между атрибутами

Выявление зависимостей между атрибутами необходимо для выполнения проектирования БД методом нормальных форм, рассматриваемого далее.

Основной способ определения наличия функциональных зависимостей – внимательный анализ семантики атрибутов. Для каждого отношения существует, но не всегда, определенное множество функциональных зависимостей между атрибутами. Причем если в некотором отношении существует одна или несколько функциональных зависимостей, можно вывести другие функциональные зависимости, существующие в этом отношении.

Пример. Пусть задано отношение R со схемой R(A1, A2, A3) и числовыми значениями, приведенными в следующей таблице:

A1	A2	A3
12	21	34
17	21	34
11	24	33
13	25	31
15	23	35
14	22	32

Априори известно, что в R существуют функциональные зависимости:

$A_1 \rightarrow A_2$ и $A_2 \rightarrow A_3$.

Анализируя это отношение, можно увидеть, что в нем существуют еще зависимости:

$A_1 \rightarrow A_3$, $A_1A_2 \rightarrow A_3$, $A_1A_2A_3 \rightarrow A_1A_2$,
 $A_1A_2 \rightarrow A_2A_3$ и т. п.

В то же время в отношении нет других функциональных зависимостей, что во введенных нами обозначениях можно отразить следующим образом:

$A_2 \rightarrow A_1$, $A_3 \rightarrow A_1$ и т. д.

Отсутствие зависимости A_1 от A_2 ($A_2 \rightarrow A_1$) объясняется тем, что одному и тому же значению атрибута A_2 (21) соответствуют разные значения атрибута A_1 (12 и 17). Другими словами, имеет место многозначность, а не функциональность.

Перечислив все существующие функциональные зависимости в отношении R, получим полное множество функциональных зависимостей, которое обозначим F^+ .

Таким образом, для последнего примера исходное множество $F = (A_1 \rightarrow A_2, A_2 \rightarrow A_3)$, а полное множество $F^+ = (A_1 \rightarrow A_2, A_2 \rightarrow A_3, A_1 \rightarrow A_3, A_1A_2 \rightarrow A_3, A_1A_2A_3 \rightarrow A_1A_2, A_1A_2 \rightarrow A_2A_3, \dots)$.

Для построения F^+ из F необходимо знать ряд правил (или аксиом) вывода одних функциональных зависимостей из других.

Существует 8 основных аксиом вывода: рефлексивности, пополнения, транзитивности, расширения, продолжения, псевдотранзитивности, объединения и декомпозиции. Перечисленные аксиомы обеспечивают получение всех ФЗ, т. е. их совокупность применительно к процедуре вывода можно считать «функционально полной». Содержание аксиом и соответствующие примеры приведены в Приложении 1.

Выявим зависимости между атрибутами отношения ПРЕПОДАВАТЕЛЬ, приведенного на рис. 5.4. При этом учтем следующее условие, которое выполняется в данном отношении: один преподаватель в одной группе может проводить один вид занятий (лекции или практические занятия).

В результате анализа отношения получаем зависимости между атрибутами, показанные на рис. 5.5.

К выделению этих ФЗ для рассматриваемого примера приводят следующие соображения.

Фамилия, имя и отчество у преподавателей факультета уникальны. Каждому преподавателю однозначно соответствует его стаж, т. е. имеет место функциональная зависимость ФИО \rightarrow Стаж. Обратное утверждение неверно, так как одинаковый стаж может быть у разных преподавателей.

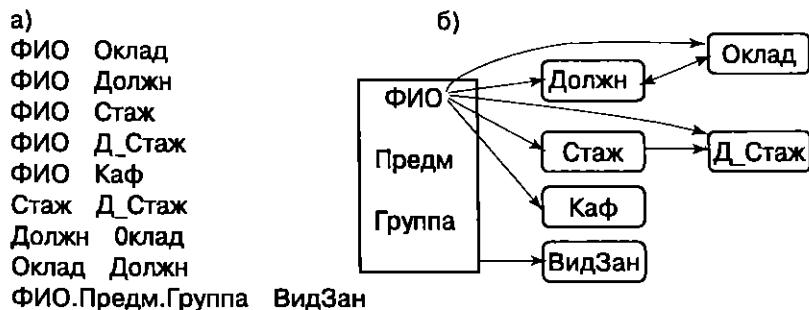


Рис. 5.5. Зависимости между атрибутами

Каждый преподаватель имеет определенную добавку за стаж, т. е. имеет место функциональная зависимость $\text{ФИО} \rightarrow \text{Д}_\text{Стаж}$, но обратная функциональная зависимость отсутствует, так как одну и ту же надбавку могут иметь несколько преподавателей.

Каждый преподаватель имеет определенную должность (преп., ст.преп., доцент, профессор), но одну и ту же должность могут иметь несколько преподавателей, т. е. имеет место функциональная зависимость $\text{ФИО} \rightarrow \text{Должн}$, а обратная функциональная зависимость отсутствует.

Каждый преподаватель является сотрудником одной и только одной кафедры. Поэтому функциональная зависимость $\text{ФИО} \rightarrow \text{Каф}$ имеет место. С другой стороны, на каждой кафедре много преподавателей, поэтому обратной функциональной зависимости нет.

Каждому преподавателю соответствует конкретный оклад, который одинаков для всех педагогов с одинаковыми должностями, что учитывается зависимостями $\text{ФИО} \rightarrow \text{Оклад}$ и $\text{Должн} \rightarrow \text{Оклад}$. Нет одинаковых окладов для разных должностей, поэтому имеет место функциональная зависимость $\text{Оклад} \rightarrow \text{Должн}$.

Один и тот же преподаватель в одной группе по разным предметам может проводить разные виды занятий. Определение вида занятий, которые проводит преподаватель, невозможно без указания предмета и группы, поэтому имеет место функциональная зависимость $\text{ФИО}, \text{Предм}, \text{Группа} \rightarrow \text{ВидЗан}$. Действительно, Петров М.И. в 256-й группе читает лекции и проводит практические занятия. Но лекции он читает по СУБД, а практику проводит по Паскалю.

Нами не были выделены зависимости между атрибутами ФИО , Предм и Группа , поскольку они образуют составной ключ и не учитываются в процессе нормализации исходного отношения.

После того, как выделены все функциональные зависимости, следует проверить их согласованность с данными исходного отношения ПРЕПОДАВАТЕЛЬ (рис. 5.4).

Например, Должн.=’преп’ и Оклад=500 всегда соответствуют друг другу во всех кортежах, т. е. подтверждается функциональная зависимость Должн. \leftrightarrow Оклад. Так же следует верифицировать и остальные функциональные зависимости, не забывая об ограниченности имеющихся в отношении данных.

Нормальные формы

Процесс проектирования БД с использованием метода нормальных форм является итерационным и заключается в последовательном переводе отношений из первой нормальной формы в нормальные формы более высокого порядка по определенным правилам. Каждая следующая нормальная форма ограничивает определенный тип функциональных зависимостей, устраивает соответствующие аномалии при выполнении операций над отношениями БД и сохраняет свойства предшествующих нормальных форм.

Выделяют следующую последовательность нормальных форм:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);
- усиленная третья нормальная форма, или нормальная форма Бойса – Кодда (БКНФ);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма (5НФ).

Первая нормальная форма. Отношение находится в 1НФ, если все его атрибуты являются простыми (имеют единственное значение). Исходное отношение строится таким образом, чтобы оно было в 1НФ.

Перевод отношения в следующую нормальную форму осуществляется методом «декомпозиции без потерь». Такая декомпозиция должна обеспечить то, что запросы (выборка данных по условию) к исходному отношению и к отношениям, получаемым в результате декомпозиции, дадут одинаковый результат.

Основной операцией метода является операция проекции. Поясним ее на примере. Предположим, что в отношении $R(A,B,C,D,E,\dots)$ устранение функциональной зависимости $C \rightarrow D$ позволит перевести его в следующую нормальную форму. Для решения этой задачи выполним декомпозицию отношения R на два новых отношения $R1(A,B,C,E,\dots)$ и $R2(C,D)$. Отношение $R2$ является проекцией отношения R на атрибуты C и D .

Исходное отношение ПРЕПОДАВАТЕЛЬ, используемое для иллюстрации метода, имеет составной ключ ФИО, Предм, Группа и находится в 1НФ, поскольку все его атрибуты простые.

В этом отношении в соответствии с рис. 5.5 б можно выделить частичную зависимость атрибутов Стаж, Д_Стаж, Каф, Должн, Оклад от ключа – указанные атрибуты находятся в функциональной зависимости от атрибута ФИО, являющегося частью составного ключа.

Эта частичная зависимость от ключа приводит к следующему:

1. В отношении присутствует явное и неявное избыточное дублирование данных, например:

- повторение сведений о стаже, должности и окладе преподавателей, проводящих занятия в нескольких группах и/или по разным предметам;
- повторение сведений об окладах для одной и той же должности или о надбавках за одинаковый стаж.

2. Следствием избыточного дублирования данных является проблема их редактирования. Например, изменение должности у преподавателя Иванова И.М. потребует просмотра всех кортежей отношения и внесения изменений в те из них, которые содержат сведения о данном преподавателе.

Часть избыточности устраняется при переводе отношения в 2НФ.

Вторая нормальная форма. Отношение находится в 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от первичного ключа (составного).

Для устранения частичной зависимости и перевода отношения в 2НФ необходимо, используя операцию проекции, разложить его на несколько отношений следующим образом:

- построить проекцию без атрибутов, находящихся в частичной функциональной зависимости от первичного ключа;
- построить проекции на части составного первичного ключа и атрибуты, зависящие от этих частей.

В результате получим два отношения R1 и R2 в 2НФ (рис. 5.6).

В отношении R1 первичный ключ является составным и состоит из атрибутов ФИО, Предм., Группа. Напомним, что данный ключ в отношении R1 получен в предположении, что каждый преподаватель в одной группе по одному предмету может либо читать лекции, либо проводить практические занятия. В отношении R2 ключ ФИО.

Исследование отношений R1 и R2 показывает, что переход к 2НФ позволил исключить явную избыточность данных в таблице R2 – повторение строк со сведениями о преподавателях. В R2 по-прежнему имеет место неявное дублирование данных.

Для дальнейшего совершенствования отношения необходимо преобразовать его в 3НФ.

Третья нормальная форма.

Определение 1. Отношение находится в 3НФ, если оно находится в 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Существует и альтернативное определение.

Определение 2. Отношение находится в 3НФ в том и только в том случае, если все неключевые атрибуты отношения взаимно независимы и полностью зависят от первичного ключа.

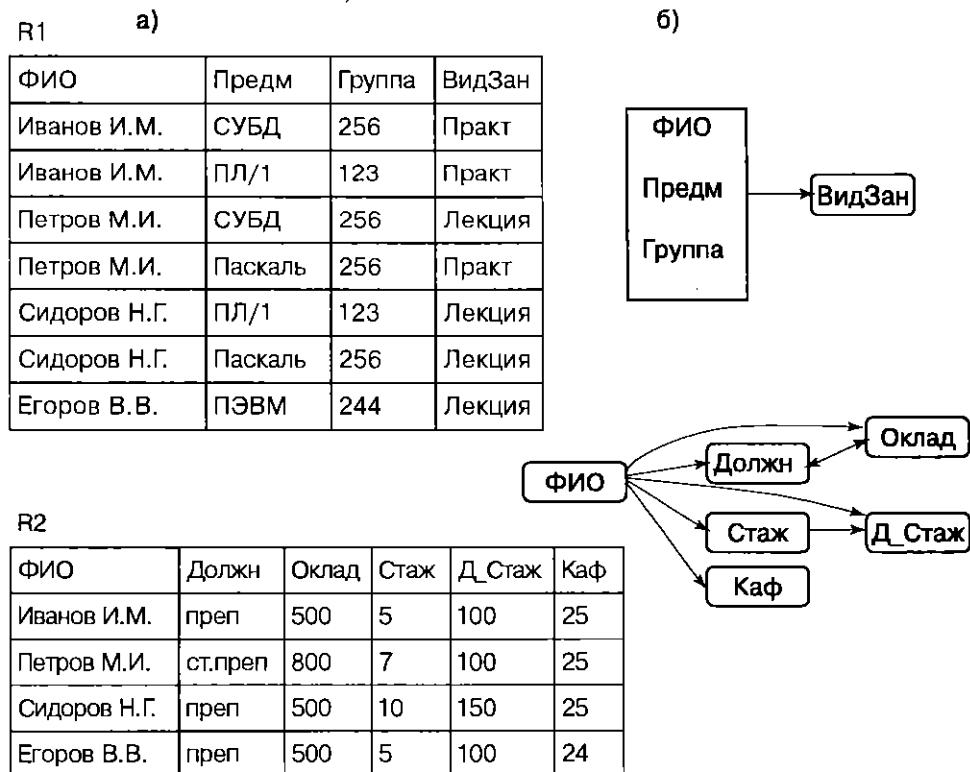


Рис. 5.6. Отношения БД в 2НФ

Доказать справедливость этого утверждения несложно. Действительно, то, что неключевые атрибуты полностью зависят от первичного ключа, означает, что данное отношение находится в форме 2НФ. Взаимная независимость атрибутов (определение приведено выше) означает отсутствие всякой зависимости между атрибутами отношения, в том числе и транзитивной зависимости между ними. Таким образом, второе определение 3НФ сводится к первому определению.

Если в отношении R1 транзитивные зависимости отсутствуют, то в отношении R2 они есть:

ФИО→Должн→Оклад,
ФИО→Оклад→Должн,
ФИО→Стаж→Д_Стаж

Транзитивные зависимости также порождают избыточное дублирование информации в отношении. Устраним их. Для этого используя операцию

проекции на атрибуты, являющиеся причиной транзитивных зависимостей, преобразуем отношение R2, получив при этом отношения R3, R4 и R5, каждое из которых находится в ЗНФ (рис. 5.7а). Графически эти отношения представлены на рис. 5.7б. Заметим, что отношение R2 можно преобразовать по-другому, а именно: в отношении R3 вместо атрибута Должн взять атрибут Оклад.

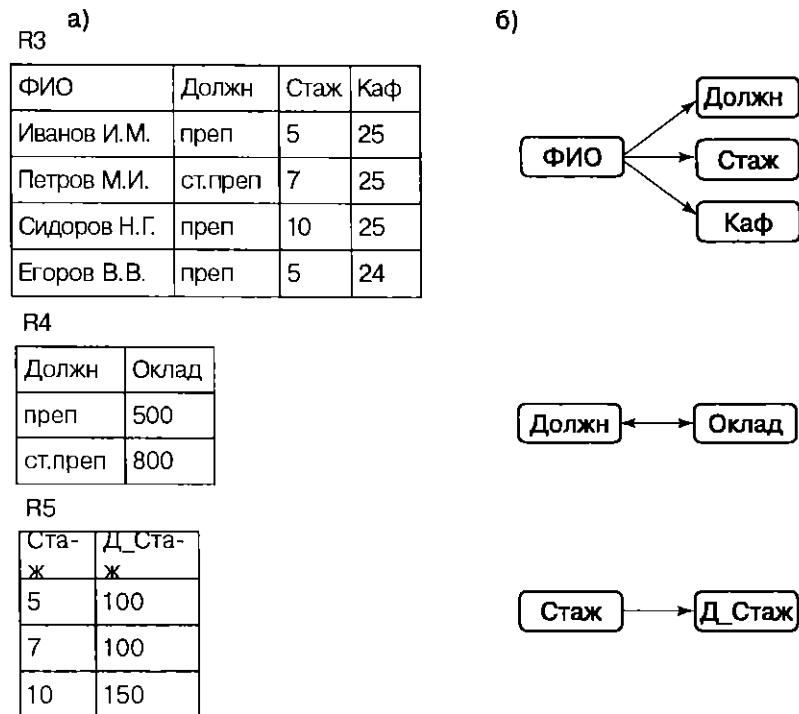


Рис. 5.7. Отношения БД в ЗНФ

На практике построение ЗНФ схем отношений в большинстве случаев является достаточным и приведением к ним процесс проектирования реляционной БД заканчивается. Действительно, приведение отношений к ЗНФ в нашем примере, привело к устраниению избыточного дублирования.

Если в отношении имеется зависимость атрибутов составного ключа от неключевых атрибутов, то необходимо перейти к усиленной ЗНФ.

Усиленная ЗНФ или нормальная форма Бойса – Кодда (БКНФ).

Отношение находится в БКНФ, если оно находится в ЗНФ и в нем отсутствуют зависимости ключей (атрибутов составного ключа) от неключевых атрибутов.

У нас подобной зависимости нет, поэтому процесс проектирования на этом заканчивается. Результатом проектирования является БД, состоящая из следующих таблиц: R1, R3, R4, R5. В полученной БД имеет место необходимое дублирование данных, но отсутствует избыточное.

Четвертая нормальная форма.

Рассмотрим пример нового отношения ПРОЕКТЫ, схема которого выглядит следующим образом: ПРОЕКТЫ (Номер_проекта, Код_сотрудника, Задание_сотрудника). Первичным ключом отношения является вся совокупность атрибутов: Номер_проекта, Код_сотрудника и Задание_сотрудника.

В отношении содержатся номера проектов, для каждого проекта – список кодов сотрудников-исполнителей, а также список заданий, предусмотренных каждым проектом. Сотрудники могут участвовать в нескольких проектах, и разные проекты могут содержать одинаковые задания. Предполагается, что каждый сотрудник, участвующий в некотором проекте, выполняет все задания по этому проекту (предположение не всегда справедливо, но желательно для нашего примера).

При такой постановке вопроса единственным возможным ключом отношения является составной атрибут Номер_проекта, Код_сотрудника, Задание_сотрудника. Он, естественно, и стал первичным ключом отношения. Отсюда следует, что отношение ПРОЕКТЫ, находится в форме БКНФ.

Пусть исходная информация в этом отношении выглядит следующим образом:

ПРОЕКТЫ

Номер_проекта	Код_сотрудника	Задание_сотрудника
001	05	1
001	05	2
001	05	3
004	02	1
004	02	2
004	03	1
004	03	2
004	05	1
004	05	2
007	06	1

Главный недостаток отношения ПРОЕКТЫ состоит в том, что при подключении/отстранении от проекта некоторого сотрудника приходится добавлять/исключать из отношения столько кортежей, сколько заданий имеется в

проекте. Внесение или исключение в отношении одного факта о некотором сотруднике требует серии элементарных операций из-за дублирования значений в кортежах.

Отсюда возникают вопросы: зачем хранить в кортежах повторяющиеся значения кодов сотрудников? Нужно ли перечислять все задания по каждому проекту, да еще для каждого сотрудника-исполнителя этого проекта? Нельзя ли информацию о привязке заданий к проектам поместить в отдельную таблицу и исключить повторения в основной таблице?

Заметим, что косвенный признак аномалии, как и ранее, – дублирование информации в таблице. Выскажем предположение, что причиной аномалии является наличие некоторой зависимости между атрибутами отношения (как увидим далее – многозначной зависимости).

Действительно, в отношении ПРОЕКТЫ существуют следующие две многозначные зависимости:

Номер_проекта \Rightarrow Код_сотрудника
 Номер_проекта \Rightarrow Задание_сотрудника

В произвольном отношении $R(A, B, C)$ может одновременно существовать многозначная зависимость $A \Rightarrow B$ и $A \Rightarrow C$. Это обстоятельство обозначим как $A \Rightarrow B \mid C$.

Дальнейшая нормализация отношений, схожих с отношением **Проекты**, основывается на следующей теореме.

Теорема Фейджина (Fagin R.). Отношение $R(A, B, C)$ можно спроектировать без потерь в отношения $R1(A, B)$ и $R2(A, C)$ в том и только том случае, когда существует зависимость $A \Rightarrow B \mid C$.

Под проектированием без потерь здесь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем *естественного соединения* полученных отношений (см. подраздел 3.6).

Поясним проектирование без потерь на примере.

Пусть имеется простейшее отношение $R(A, B, C)$, имеющее вид:

R		
A	B	C
К	15	1
К	15	2
Л	10	1
М	20	1
М	20	2
М	20	3

Построим проекции R1 и R2 на атрибуты A, B и A, C соответственно. Они будут выглядеть так:

R1

A	B
К	15
Л	10
М	20

R2

A	C
К	1
К	2
Л	1
М	1
М	2
М	3

Результатом *операции соединения* бинарных отношений R1(A, B) и R2(A, C) по атрибуту A является тернарное отношение с атрибутами A, B и C, кортежи которого получаются путем связывания отношений R1 и R2 по типу 1:M на основе совпадения значений атрибута A (подраздел 3.3).

Так, связывание кортежей (к, 15) и {(к, 1), (к, 2)} дает кортежи {(к, 15, 1), (к, 15, 2)}.

Нетрудно видеть, что связывание R1(A, B) и R2(A, C) в точности порождает исходное отношение R(A, B, C). В отношении R нет лишних кортежей, нет и потерь.

Определение четвертой нормальной формы. Отношение R находится в четвертой нормальной форме (4НФ) в том и только в том случае, когда существует многозначная зависимость A⇒B, а все остальные атрибуты R функционально зависят от A.

Приведенное выше отношение ПРОЕКТЫ можно представить в виде двух отношений: ПРОЕКТЫ-СОТРУДНИКИ и ПРОЕКТЫ-ЗАДАНИЯ. Структура этих отношений и содержимое соответствующих таблиц выглядит следующим образом:

ПРОЕКТЫ-СОТРУДНИКИ (Номер_проекта, Код_сотрудника).

Первичный ключ отношения: Номер_проекта, Код_сотрудника.

ПРОЕКТЫ-СОТРУДНИКИ

Номер_проекта	Код_сотрудника
001	05
004	02
004	03
004	05
007	06

ПРОЕКТЫ-ЗАДАНИЯ (Номер_проекта, Задание_сотрудника).

Первичный ключ отношения: Номер_проекта, Задание_сотрудника.

ПРОЕКТЫ-ЗАДАНИЯ

Номер_проекта	Задание_сотрудника
001	1
001	2
001	3
004	1
004	2
007	1

Как легко увидеть, оба этих отношения находятся в 4НФ и свободны от замечанных недостатков. Дублирование значений атрибутов кодов сотрудников пропало. Попробуйте самостоятельно соединить эти отношения и убедиться в том, что в точности получится отношение ПРОЕКТЫ.

В общем случае не всякое отношение можно восстановить к исходному. В нашем случае восстановление возможно потому, что каждый сотрудник, участвующий в некотором проекте, выполнял все задания по этому проекту (именно это укладывается в принцип 1:М соединения отношений). Самые же сотрудники участвовали в нескольких проектах, и разные проекты могли содержать одинаковые задания.

Пятая нормальная форма.

Результатом нормализации всех предыдущих схем отношений были два новых отношения. Иногда это сделать не удается, либо получаемые отношения заранее имеют нежелательные свойства. В этом случае выполняют декомпозицию исходного отношения на отношения, количество которых превышает два.

Рассмотрим отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ, которое имеет заголовок СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ (Код_сотрудника, Код_отдела, Номер_проекта). Первичный ключ отношения включает все атрибуты: Код_сотрудника, Код_отдела и Номер_проекта. Пусть в этом отношении один сотрудник может работать в нескольких отделах, причем в каждом отделе он может принимать участие в нескольких проектах. В одном отделе могут работать несколько сотрудников, но каждый проект выполняет только один сотрудник. Функциональных и многозначных зависимостей между атрибутами не существует.

Это отношение является частью базы данных вымышленного научного подразделения НИЧАВО - Научно-Исследовательского Института Чародейства и Волшебства из повести А. и Б. Стругацких «Понедельник начинается в субботу». Коды отделов здесь обозначают: АД – Администрация,

СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ

Код_сотрудника	Код_отдела	Номер_проекта
01	РД	036
02	АД	004
03	УП	004
04	АД	019
05	ЛС	001
05	ЛС	004
06	УП	007
08	ВЦ	013
09	ВЦ	014
10	СЖ	013

ВЦ – Вычислительный центр, ЛС – Линейного счастья, РД – Родильный дом, СЖ – Смысла жизни, УП – Универсальных превращений.

Исходя из структуры отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ можно заключить, что оно находится в форме 4НФ. Тем не менее в отношении могут быть аномалии, связанные с возможностью повторения значений атрибутов в нескольких кортежах. Например, то, что сотрудник может работать в нескольких отделах, при увольнении сотрудника требует отыскания и последующего удаления из исходной таблицы нескольких записей.

Введем определение *зависимости соединения*. Отношение $R(X, Y, \dots, Z)$ удовлетворяет *зависимости соединения*, которую обозначим как $*(X, Y, \dots, Z)$, в том и только в том случае, если R восстанавливается без потерь путем соединения своих проекций на X, Y, \dots, Z . Зависимость соединения является обобщением функциональной и многозначной зависимостей.

Определение пятой нормальной формы. Отношение R находится в 5НФ (или нормальной форме проекции-соединения – PJ/NF) в том и только том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R .

Образуем составные атрибуты отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ:

СО={ Код_сотрудника, Код_отдела }
 СП={ Код_сотрудника, Номер_проекта }
 ОП={ Код_отдела, Номер_проекта }.

Покажем, что если отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ спроектировать на составные атрибуты СО, СП и ОП, то соединение этих про-

екций дает исходное отношение. Это значит, что в нашем отношении существовала зависимость соединения *(СО, СП, ОП). Проекции на составные атрибуты назовем соответственно СОТРУДНИКИ-ОТДЕЛЫ, СОТРУДНИКИ-ПРОЕКТЫ и ОТДЕЛЫ-ПРОЕКТЫ.

Ранее мы выполняли соединение двух проекций и сразу получали искомый результат. Для восстановления отношения из трех (или нескольких) проекций надо получить все попарные соединения (так как информация о том, какое из них «лучше», отсутствует), над которыми затем выполнить операцию пересечения множеств. Проверим, так ли это.

СОТРУДНИКИ-ОТДЕЛЫ

Код_сотрудника	Код_отдела
01	РД
02	АД
03	УП
04	АД
05	ЛС
05	ЛС
06	УП
08	ВЦ
09	ВЦ
10	СЖ

СОТРУДНИКИ-ПРОЕКТЫ

Код_сотрудника	Номер_проекта
01	036
02	004
03	004
04	019
05	001
05	004
06	007
08	013
09	014
10	013

ОТДЕЛЫ-ПРОЕКТЫ

Код_отдела	Номер_проекта
АД	004
АД	019
ВЦ	013
ВЦ	014
ЛС	001
ЛС	004
СЖ	013
РД	036
УП	004
УП	007

Получим попарные соединения трех приведенных выше отношений, которые будут иметь вид:

*(СО, СП)

Код_сотрудника	Код_отдела	Номер_проекта
01	РД	036
02	АД	004
03	УП	004
04	АД	019
05	ЛС	001

05	ЛС	004
06	УП	007
08	ВЦ	013
09	ВЦ	014
10	СЖ	013

*(СО, ОП)

Код_сотрудника	Код_отдела	Номер_проекта
01	РД	036
02	АД	004
02	АД	019
03	УП	004
03	УП	007
04	АД	004
04	АД	019
05	ЛС	001
05	ЛС	004
06	УП	004
06	УП	007
08	ВЦ	013
08	ВЦ	014
09	ВЦ	013
09	ВЦ	014
10	СЖ	013

*(СП, ОП)

Код_сотрудника	Код_отдела	Номер_проекта
01	РД	036
02	АД	004
02	ЛС	004
02	УП	004
03	АД	004
03	ЛС	004
03	УП	004
04	АД	019
05	ЛС	001
05	АД	004
05	ЛС	004
05	УП	004
06	УП	004
06	УП	007
08	ВЦ	013
08	ВЦ	014
09	ВЦ	013
09	ВЦ	014
10	СЖ	013

Нетрудно увидеть, что пересечение всех отношений дает исходное отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ.

Замечание.

Существуют и другие способы восстановления исходного отношения из его проекций. Так, для восстановления отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ можно соединить отношения СОТРУДНИКИ-ОТДЕЛЫ и СОТРУДНИКИ-ПРОЕКТЫ по атрибуту Код_сотрудника, после чего полученное отношение соединить с отношением ОТДЕЛЫ-ПРОЕКТЫ по составному атрибуту (Код_отдела, Номер_проекта).

Отношения СОТРУДНИКИ-ОТДЕЛЫ, СОТРУДНИКИ-ПРОЕКТЫ и ОТДЕЛЫ-ПРОЕКТЫ находятся в ЗНФ. Эта форма является последней из известных. Условия ее получения довольно нетривиальны и поэтому она почти не используется на практике. Более того, она имеет определенные недостатки!

Предположим, необходимо узнать, где и какие проекты исполняет сотрудник с кодом 02. Для этого в отношении СОТРУДНИКИ-ОТДЕЛЫ найдем, что сотрудник с кодом 02 работает в отделе АД, а из отношения ОТДЕЛЫ-ПРОЕКТЫ найдем, что в отделе АД выполняются проекты 004 и 019. А это значит, что сотрудник 02 должен выполнять проекты 004 и 019. Увы, информация о том, что сотрудник с кодом 02 выполняет проект 019, ошибочна (см. исходное отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ).

Такие противоречия можно устранить только путем совместного рассмотрения всех проекций основного отношения. Именно поэтому после соединения проекций и выполнялась операция их пересечения. Безусловно, это – недостаток. Отметим, что наличие недостатков не требует обязательного отказа от определенных видов нормальных форм. Надо учитывать недостатки и условия их проявления. В некоторых постановках задач недостатки не проявляются.

На практике обычно ограничиваются структурой БД, соответствующей ЗНФ или БКНФ. Поэтому процесс нормализации отношений методом нормальных форм предполагает последовательное удаление из исходного отношения следующих межатрибутных зависимостей:

- частичных зависимостей неключевых атрибутов от ключа (удовлетворение требований 2НФ);
- транзитивных зависимостей неключевых атрибутов от ключа (удовлетворение требований ЗНФ);
- зависимости ключей (атрибутов составных ключей) от неключевых атрибутов (удовлетворение требований БКНФ).

Кроме метода нормальных форм Кодда, используемого для проектирования небольших БД, применяют и другие методы, например, метод ER-диаграмм (метод «Сущность-связь»). Этот метод используется при проектировании больших БД, на нем основан ряд средств проектирования БД. Метод ER-диаграмм рассматривается в следующем разделе.

На последнем этапе метода ER-диаграмм отношения, полученные в результате проектирования, проверяются на принадлежность их к БКНФ. Этот этап может выполняться уже с использованием метода нормальных форм. После завершения проектирования создается БД с помощью СУБД.

5.3. Рекомендации по разработке структур

В качестве обобщения материала предыдущего подраздела приведем наиболее важные рекомендации, неучет которых может привести к аномалиям при обработке данных в базах. Остановимся на двух вопросах: исходя из каких соображений нужно создавать отношения (таблицы) и каким образом следует их связывать.

Какими должны быть таблицы сущностей

Основное правило при создании таблиц сущностей – это «каждой сущности – отдельную таблицу» (как в популярном лозунге: «каждой семье – отдельную квартиру»).

Поля таблиц сущностей могут быть двух видов: *ключевые* и *неключевые*. Введение ключей в таблице практически во всех реляционных СУБД позволяет обеспечить уникальность значений в записях таблицы по ключу, ускорить обработку записей таблицы, а также выполнить автоматическую сортировку записей по значениям в ключевых полях.

Обычно достаточно определения *простого* ключа, реже – вводят *составной* ключ. Таблицей с составным ключом может быть, например, таблица хранения списка сотрудников (фамилия, имя и отчество), в котором встречаются однофамильцы. В некоторых СУБД пользователям предлагается определить автоматически создаваемое ключевое поле нумерации (в Access – это поле типа «счетчик»), которое упрощает решение проблемы уникальности записей таблицы.

Иногда в таблицах сущностей имеются поля описания свойств или характеристик объектов. Если в таблице есть значительное число повторений по этим полям и эта информация имеет существенный объем, то лучше их выделить в отдельную таблицу (придерживаясь правила: «каждой сущности – отдельную таблицу»). Тем более, следует образовать дополнительную таблицу, если свойства взаимосвязаны.

В более общем виде последние рекомендации можно сформулировать так: информацию о сущностях следует представить таким образом, чтобы неключевые поля в таблицах были взаимно независимыми и полностью зависели от ключа (см. определение третьей нормальной формы).

В процессе обработки таблиц сущностей надо иметь в виду следующее. Новую сущность легко добавить и изменить, но при удалении следует уничтожить все ссылки на нее из таблиц связей, иначе таблицы связей будут некорректными. Многие современные СУБД блокируют некорректные действия в подобных операциях.

Организация связи сущностей

Записи таблицы связей предназначены для отображения связей между сущностями, информации о которых находится в соответствующих таблицах сущностей.

Обычно одна таблица связей описывает взаимосвязь двух сущностей. Поскольку таблицы сущностей в простейшем случае имеют по одному ключевому полю, то таблица связей двух таблиц для обеспечения уникальности за-

писей о связях должна иметь два ключа. Можно создать таблицу связей, как и таблицу объектов, и без ключей, но тогда функции контроля за уникальностью записей ложатся на пользователя.

Более сложные связи (не бинарные) следует сводить к бинарным. Для описания взаимосвязей N объектов требуется $N-1$ таблиц связей. Транзитивных связей не должно быть. Избыток связей приводит к противоречиям (см. пример отношений СОТРУДНИКИ-ОТДЕЛЫ, СОТРУДНИКИ-ПРОЕКТЫ и ОТДЕЛЫ-ПРОЕКТЫ предыдущего подраздела).

Не следует включать в таблицы связей характеристики сущностей, иначе неизбежны аномалии. Их лучше хранить в отдельных таблицах сущностей.

С помощью таблиц связей можно описывать и несколько специфичный вид связи – линейную связь, или слабую связь. Примером линейной связи можно считать отношение принадлежности сущностей некоторой другой сущности более высокого порядка (системы, состоящие из узлов; лекарства, состоящие из компонентов; сплавы металлов и т. д.). В этом случае для описания связей достаточно одной таблицы связей.

При работе с таблицами связей следует иметь в виду, что любая запись из таблицы связей легко может быть удалена, поскольку сущности некоторое время могут обойтись и без связей. При добавлении или изменении содержимого записей таблицы надо контролировать правильность ссылок на существующие объекты, так как связь без объектов существовать не может. Большинство современных СУБД контролируют правильность ссылок на объекты.

5.4. Обеспечение целостности

Под *целостностью* понимают свойство базы данных, означающее, что она содержит полную, непротиворечивую и адекватно отражающую предметную область информацию.

Различают *физическую* и *логическую* целостность. Физическая целостность означает наличие физического доступа к данным и то, что данные не утрачены. Логическая целостность означает отсутствие логических ошибок в базе данных, к которым относятся нарушение структуры БД или ее объектов, удаление или изменение установленных связей между объектами и т. д. В дальнейшем речь будет вести о логической целостности.

Поддержание целостности БД включает проверку (контроль) целостности и ее восстановление в случае обнаружения противоречий в базе. Целостное состояние БД задается с помощью *ограничений целостности* в виде условий, которым должны удовлетворять хранимые в базе данные.

Среди ограничений целостности можно выделить два основных типа ограничений: *ограничения значений* атрибутов отношений и *структурные ограничения* на кортежи отношений.

Примером **ограничений значений** атрибутов отношений является требование недопустимости пустых или повторяющихся значений в атрибутах, а также контроль принадлежности значений атрибутов заданному диапазону. Так, в записях отношений о кадрах значения атрибута *Дата_рождения* не могут превышать значений атрибута *Дата_приема*.

Наиболее гибким средством реализации контроля значений атрибутов являются *хранимые процедуры и триггеры*, имеющиеся в некоторых СУБД.

Структурные ограничения определяют требования *целостности сущностей и целостности ссылок*. Каждому экземпляру сущности, представленному в отношении, соответствует только один его кортеж. Требование *целостности сущностей* состоит в том, что любой кортеж отношения должен быть отличим от любого другого кортежа этого отношения, т. е., иными словами, любое отношение должно обладать первичным ключом.

Формулировка требования целостности ссылок тесно связана с понятием *внешнего ключа*. Напомним, что внешние ключи служат для связи отношений (таблиц БД) между собой. При этом атрибут одного отношения (родительского) называется *внешним ключом* данного отношения, если он является *первичным ключом* другого отношения (дочернего). Говорят, что отношение, в котором определен внешний ключ, ссылается на отношение, в котором этот же атрибут является первичным ключом.

Требование целостности ссылок состоит в том, что для каждого значения внешнего ключа родительской таблицы должна найтись строка в дочерней таблице с таким же значением первичного ключа. Например, если в отношении R1 (рис. 5.8) содержатся сведения о сотрудниках кафедры, а атрибут этого отношения *Должн* является первичным ключом отношения R2, то в этом отношении для каждой должности из R1 должна быть строка с соответствующим ей окладом.

Во многих современных СУБД имеются средства обеспечения контроля целостности БД.

Родительская таблица				Дочерняя таблица	
R1				R2	
ФИО	Должн	Каф	Стаж	Должн	Оклад
Иванов И.М.	преп	25	5		
Петров М.И.	ст.преп	25	7		
Сидоров Н.Г.	преп	25	10		
Егоров В.В.	преп	24	5		

Внешний
ключ

Рис. 5.8. Связь отношений с помощью внешнего ключа

Контрольные вопросы и задания

1. Назовите подходы к проектированию структур данных.
2. В чем состоит избыточное и неизбыточное дублирование данных?
3. Назовите и охарактеризуйте основные виды аномалий.
4. Как формируется исходное отношение при проектировании БД?
5. Приведите примеры явной и неявной избыточности.
6. Назовите основные виды зависимостей между атрибутами отношений.
7. Приведите примеры функциональной и частичной функциональной зависимости.
8. Приведите примеры отношений с зависимыми атрибутами.
9. Охарактеризуйте нормальные формы.
10. Дайте определение первой нормальной формы.
11. Дайте определение второй нормальной формы.
12. Дайте определение третьей нормальной формы.
13. Дайте определение усиленной третьей нормальной формы.
14. Поясните на примере используемых в разделе таблиц требования 4НФ.
15. Поясните на примере используемых в разделе таблиц требования 5НФ.
16. Сформулируйте основное правило создания таблиц сущностей.
17. Назовите рекомендации по организации связи сущностей.
18. Дайте определение физической и логической целостности БД.
19. Приведите примеры ограничений значений и структурных ограничений.
20. Поясните понятия внешнего и первичного ключей таблиц.
21. Разработайте схему БД, позволяющую просматривать и редактировать информацию об автомобильном парке организации. БД должна содержать сведения о водителях машин (категория транспортных средств, водительский стаж, закрепленные автомобили и т. д.), а также автомобилях автопарка (марка, год выпуска, техническое состояние и т. д.). Предложите и обоснуйте выбор структур таблиц, их взаимосвязь и укажите вид нормальных форм таблиц.

Литература

1. *Бородаев В. А., Кустюк В. Н. Банки и базы данных. Уч. пособие.* Л.: ВИКИ, 1989.
2. *Дейт К. Дж. Введение в системы баз данных / Пер. с англ. – 6-е изд. – К.: Диалектика, 1998.*
3. *Кузнецов С. Д. Введение в СУБД: часть 4 // Системы Управления Базами Данных, № 4, 1995. – С. 114–122.*
4. *Основы современных компьютерных технологий: Учебник / Под ред. проф. Хомоненко А. Д. Авторы: Брякалов Г. А., Войцеховский С. В., Воробьев Е. Г., Гофман В. Э., Гридин В. В., Дрюков Ю. П., Замула А. А., Захаров А. И., Компаниец Р. И., Липецких А. Г., Рыжиков Ю. И., Хомоненко А. Д., Цыганков В. М. – СПб: КОРОНА прнт, 2005. – 672 с.*
5. *Плакс М. Б. Основы программирования на языке Clipper. Справочное издание. СПб.: А/О ДиалогИнвест, ТОО ЛенДиасофт.*

6. Метод сущность-связь

Метод сущность-связь называют также методом «ER-диаграмм»: во-первых, ER – аббревиатура от слов *Essence* (*сущность*) и *Relation* (*связь*), во-вторых, метод основан на использовании диаграмм, называемых соответственно диаграммами ER-экземпляров и диаграммами ER-типа.

6.1. Основные понятия метода

Основными понятиями метода сущность-связь являются следующие:

- сущность,
- атрибут сущности,
- ключ сущности,
- связь между сущностями,
- степень связи,
- класс принадлежности экземпляров сущности,
- диаграммы ER-экземпляров,
- диаграммы ER-типа.

Сущность представляет собой объект, информация о котором хранится в БД. Экземпляры сущности отличаются друг от друга и однозначно идентифицируются. Названиями сущностей являются, как правило, *существительные*, например: ПРЕПОДАВАТЕЛЬ, ДИСЦИПЛИНА, КАФЕДРА, ГРУППА.

Атрибут представляет собой свойство сущности. Это понятие аналогично понятию атрибута в отношении. Так, атрибутами сущности ПРЕПОДАВАТЕЛЬ может быть его Фамилия, Должность, Стаж (преподавательский) и т. д.

Ключ сущности – атрибут или набор атрибутов, используемый для идентификации экземпляра сущности. Как видно из определения, понятие ключа сущности аналогично понятию ключа отношения.

Связь двух или более сущностей – предполагает зависимость между атрибутами этих сущностей. Название связи обычно представляется глаголом. Примерами связей между сущностями являются следующие: ПРЕПОДАВАТЕЛЬ ВЕДЕТ ДИСЦИПЛИНУ (Иванов ВЕДЕТ «Базы данных»), ПРЕПОДАВАТЕЛЬ ПРЕПОДАЕТ-В ГРУППЕ (Иванов ПРЕПОДАЕТ-В 256 группе), ПРЕПОДАВАТЕЛЬ РАБОТАЕТ-НА КАФЕДРЕ (Иванов РАБОТАЕТ-НА кафедре).

Приведенные определения сущности и связи не полностью формализованы, но приемлемы для практики. Следует иметь в виду, что в результате проектирования могут быть получены несколько вариантов одной БД. Так, два разных проектировщика, рассматривая одну и ту же проблему с разных точек зрения, могут получить различные наборы сущностей и связей. При этом оба

варианта могут быть рабочими, а выбор лучшего из них будет результатом личных предпочтений.

С целью повышения наглядности и удобства проектирования для представления сущностей, экземпляров сущностей и связей между ними используются следующие графические средства:

- диаграммы ER-экземпляров,
- диаграммы ER-типа, или ER-диаграммы.

На рис. 6.1 приведена диаграмма ER-экземпляров для сущностей ПРЕПОДАВАТЕЛЬ и ДИСЦИПЛИНА со связью ВЕДЕТ.

ПРЕПОДАВАТЕЛЬ	ВЕДЕТ	ДИСЦИПЛИНА
ИВАНОВ И.М.	●	СУБД
ПЕТРОВ М.И.	●	ПЛ/1
СИДОРОВ Н.Г.	●	Паскаль
ЕГОРОВ В.В.	●	Алгол
КОЗЛОВ А.С.	●	Фортран

Рис. 6.1. Диаграмма ER-экземпляров

Диаграмма ER-экземпляров показывает, какую конкретно дисциплину (СУБД, ПЛ/1 и т.д.) ведет каждый из преподавателей. На рис. 6.2 представлена диаграмма ER-типа, соответствующая рассмотренной диаграмме ER-экземпляров.



Рис. 6.2. Диаграмма ER-типа

На начальном этапе проектирования БД выделяются атрибуты, составляющие ключи сущностей.

На основе анализа диаграмм ER-типа формируются отношения проектируемой БД. При этом учитывается *степень связи сущностей* и *класс их принадлежности*, которые, в свою очередь, определяются на основе анализа диаграмм ER-экземпляров соответствующих сущностей.

Степень связи является характеристикой связи между сущностями, которая может быть типа: 1:1, 1:M, M:1, M:M.

Класс принадлежности (КП) сущности может быть: *обязательным* и *необязательным*.

Класс принадлежности сущности является *обязательным*, если все экземпляры этой сущности обязательно участвуют в рассматриваемой связи, в противном случае класс принадлежности сущности является *необязательным*.

Варьируя классом принадлежности сущностей для каждого из названных типов связи, можно получить несколько вариантов диаграмм ER-типа. Рассмотрим примеры некоторых из них.

Пример 1. Связи типа 1:1 и необязательный класс принадлежности.

В приведенной на рис. 6.1 диаграмме степень связи между сущностями 1:1, а класс принадлежности обеих сущностей необязательный. Действительно, из рисунка видно следующее:

- каждый преподаватель ведет не более одной дисциплины, а каждая дисциплина ведется не более чем одним преподавателем (степень связи 1:1);
- некоторые преподаватели не ведут ни одной дисциплины и имеются дисциплины, которые не ведет ни один из преподавателей (класс принадлежности обеих сущностей необязательный).

Пример 2. Связи типа 1:1 и обязательный класс принадлежности.

На рис. 6.3 приведены диаграммы, у которых степень связи между сущностями 1:1, а класс принадлежности обеих сущностей обязательный.

В этом случае каждый преподаватель ведет одну дисциплину и каждая дисциплина ведется одним преподавателем.

а) ER-экземпляров

ПРЕПОДАВАТЕЛЬ	ВЕДЕТ	ДИСЦИПЛИНА
ИВАНОВ И.М.	●	СУБД
ПЕТРОВ М.И.	●	ПЛ/1
СИДОРОВ Н.Г.	●	Паскаль
ЕГОРОВ В.В.	●	Алгол
КОЗЛОВ А.С.	●	Фортран

б) ER-типов



Рис. 6.3. Диаграммы для связи 1:1 и обязательным КП обеих сущностей

Возможны два промежуточных варианта с необязательным классом принадлежности одной из сущностей.

Замечания.

- На диаграммах ER-типа *обязательное* участие в связи экземпляров сущности отмечается блоком с точкой внутри, смежным с блоком этой сущности (рис. 6.36).
- При *необязательном* участии экземпляров сущности в связи дополнительный блок к блоку сущности не пристраивается, а точка размещается на линии связи (рис. 6.2).
- Символы на линии связи указывают на степень связи.

Под каждым блоком, соответствующим некоторой сущности, указывается ее ключ, выделяемый подчеркиванием. Многоточие за ключевыми атрибутами означает, что возможны другие атрибуты сущности, но ни один из них не может быть частью ее ключа. Эти атрибуты выявляются после формирования отношений.

На практике степень связи и класс принадлежности сущностей при проектировании БД определяется спецификой предметной области. Рассмотрим примеры вариантов со степенью связи 1:M или M:1.

Пример 3. Связи типа 1:M.

Каждый преподаватель может вести несколько дисциплин, но каждая дисциплина ведется одним преподавателем.

Пример 4. Связи типа M:1.

Каждый преподаватель может вести одну дисциплину, но каждую дисциплину могут вести несколько преподавателей.

Примеры с типом связи 1:M или M:1 могут иметь ряд вариантов, отличающихся классом принадлежности одной или обеих сущностей. Обозначим обязательный класс принадлежности символом «О», а необязательный – символом «Н», тогда варианты для связи типа 1:M условно можно представить как: О-О, О-Н, Н-О, Н-Н. Для связи типа M:1 также имеются 4 аналогичных варианта.

Пример 5. Связи типа 1:M вариант Н-О.

Каждый преподаватель может вести несколько дисциплин или ни одной, но каждая дисциплина ведется одним преподавателем (рис. 6.4).

По аналогии легко составить диаграммы и для остальных вариантов.

Пример 6. Связи типа M:M.

Каждый преподаватель может вести несколько дисциплин, а каждая дисциплина может вестись несколькими преподавателями.

Как и в случае других типов связей, для связи типа M:M возможны 4 варианта, отличающиеся классом принадлежности сущностей.

Пример 7. Связи типа M:M и вариант класса принадлежности О-Н.

Допустим, что каждый преподаватель ведет не менее одной дисциплины, а дисциплина может вестись более чем одним преподавателем, есть и такие дисциплины, которые никто не ведет. Соответствующие этому случаю диаграммы приведены на рис. 6.5.

а) ER-экземпляров

ПРЕПОДАВАТЕЛЬ	ВЕДЕТ	ДИСЦИПЛИНА
		СУБД
ИВАНОВ И.М.	●	ПЛ/1
ПЕТРОВ М.И.	●	Паскаль
СИДОРОВ Н.Г.	●	Алгол
ЕГОРОВ В.В.	●	Фортран
КОЗЛОВ А.С.	●	C++
		JAVA

б) ER-типов



Рис. 6.4. Диаграммы для связи типа 1:М варианта Н-О

Выявление сущностей и связей между ними, а также формирование на их основе диаграмм ER-типа выполняется на начальных этапах метода *сущность-связь*. Рассмотрим этапы реализации метода.

а) ER-экземпляров

ПРЕПОДАВАТЕЛЬ	ВЕДЕТ	ДИСЦИПЛИНА
		СУБД
ИВАНОВ И.М.	●	ПЛ/1
ПЕТРОВ М.И.	●	Паскаль
СИДОРОВ Н.Г.	●	Алгол
ЕГОРОВ В.В.	●	Фортран
КОЗЛОВ А.С.	●	C++
		JAVA

б) ER-типов



Рис. 6.5. Диаграммы для связи типа М : М и варианта О-Н

6.2. Этапы проектирования

Процесс проектирования базы данных является итерационным – допускающим возврат к предыдущим этапам для пересмотра ранее принятых решений и включает следующие этапы:

1. Выделение сущностей и связей между ними.
2. Построение диаграмм ER-типа с учетом всех сущностей и их связей.
3. Формирование набора предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения и использованием диаграмм ER-типа.
4. Добавление неключевых атрибутов в отношения.
5. Приведение предварительных отношений к нормальной форме Бойса – Кодда, например, с помощью метода нормальных форм.
6. Пересмотр ER-диаграмм в следующих случаях:
 - некоторые отношения не приводятся к нормальной форме Бойса – Кодда;
 - некоторым атрибутам не находится логически обоснованных мест в предварительных отношениях.

После преобразования ER-диаграмм осуществляется повторное выполнение предыдущих этапов проектирования (возврат к этапу 1).

Одним из узловых этапов проектирования является этап формирования отношений. Рассмотрим процесс формирования предварительных отношений, составляющих первичный вариант схемы БД.

В рассмотренных выше примерах связь ВЕДЕТ всегда соединяет две сущности и поэтому является *бинарной*. Сформулированные ниже правила формирования отношений из диаграмм ER-типа распространяются именно на бинарные связи. Поэтому, когда речь идет о связях, слово «бинарные» далее опускается.

6.3. Правила формирования отношений

Правила формирования отношений основываются на учете следующего:

- степени связи между сущностями (1:1, 1:M, M:1, M:M);
- класса принадлежности экземпляров сущностей (обязательный и необязательный).

Рассмотрим формулировки шести правил формирования отношений на основе диаграмм ER-типа.

Формирование отношений для связи 1:1

Правило 1. Если степень бинарной связи 1:1 и класс принадлежности обеих сущностей обязательный, то формируется одно отношение. Первичным ключом этого отношения может быть ключ любой из двух сущностей.

На рис. 6.6 приведены диаграмма ER-типа и отношение, сформированное по правилу 1 на ее основе.

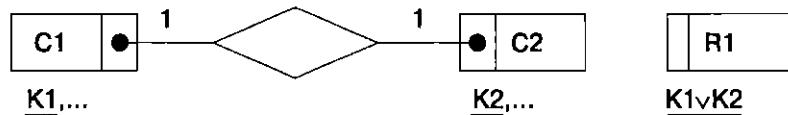


Рис. 6.6. Диаграмма и отношения для правила 1

На рисунке используются следующие обозначения:

C_1, C_2 – сущности 1 и 2;

K_1, K_2 – ключи первой и второй сущности соответственно;

R_1 – отношение 1, сформированное на основе первой и второй сущностей;

$K_1 \vee K_2, \dots$ означает, что ключом сформированного отношения может быть либо K_1 , либо K_2 .

Это и другие правила будем проверять, рассматривая различные варианты связи ПРЕПОДАВАТЕЛЬ ВЕДЕТ ДИСЦИПЛИНУ. Пусть сущность ПРЕПОДАВАТЕЛЬ характеризуется атрибутами НП (идентификационный номер преподавателя), ФИО (фамилия, имя и отчество), Стаж (стаж преподавателя). Сущность ДИСЦИПЛИНА характеризуется соответственно атрибутами КД (код дисциплины), Часы (часы, отводимые на дисциплину). Тогда схема отношения, содержащего информацию об обеих сущностях, и само отношение для случая, когда степень связи равна 1:1, а КП обязательный для всех сущностей, могут иметь вид, показанный на рис. 6.7.

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА(НП,ФИО,Стаж,КД,Часы)

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

НП	ФИО	Стаж	КД	Часы
П1	Иванов И.М.	5	K1	62
П2	Петров М.И.	7	K2	74
П3	Сидоров Н.Г.	10	K3	102
П4	Егоров В.В.	5	K4	80

Рис. 6.7. Полученные по правилу 1 схема и отношение

Сформированное отношение содержит полную информацию о преподавателях, дисциплинах и о том, как они связаны между собой. Так, преподаватель Иванов ведет только дисциплину с кодом K1, а дисциплина K1 ведется только Ивановым (связь 1:1). В этом отношении отсутствуют пустые поля (КП обязательный для всех сущностей), т. к. нет преподавателей, которые бы что-то не вели, и нет дисциплин, которые никто не ведет. Таким

образом, одного отношения в данном случае достаточно. В качестве первичного ключа может быть выбран ключ первого отношения НП или ключ второго отношения КД.

Правило 2. Если степень связи 1:1 и класс принадлежности одной сущности обязательный, а второй – необязательный, то под каждую из сущностей формируется по отношению с первичными ключами, являющимися ключами соответствующих сущностей. Далее к отношению, сущность которого имеет обязательный КП, добавляется в качестве атрибута ключ сущности с необязательным КП.

На рис. 6.8 приведены диаграмма ER-типа и отношения, сформированные по правилу 2 на ее основе.

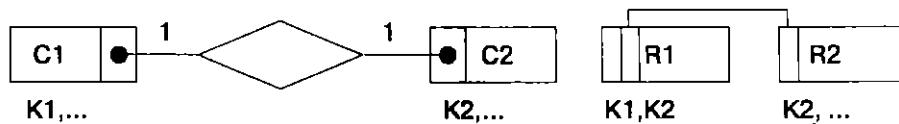


Рис. 6.8. Диаграмма и отношения для правила 2

Чтобы убедиться в справедливости правила, рассмотрим следующий пример. На рис. 6.9 приведено исходное отношение, содержащее информацию о преподавателях и дисциплинах. Оно представляет вариант, в котором класс сущности ПРЕПОДАВАТЕЛЬ является обязательным, а сущности ДИСЦИПЛИНА – необязательным. При этом пробелы «--» (пустые поля) присутствуют во всех кортежах с информацией о дисциплинах, которые не ведутся ни одним из преподавателей.

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

НП	ФИО	Стаж	КД	Часы
П1	Иванов И.М.	5	К1	62
П2	Петров М.И.	7	К2	74
П3	Сидоров Н.Г.	10	К3	102
---	---	---	К4	80

Рис. 6.9. Исходное отношение

Избежать этой ситуации можно, применив правило 2, в соответствии с которым, выделяются два отношения, приведенные на рис. 6.10.

ПРЕПОДАВАТЕЛЬ(НП, ФИО, Стаж, КД)

ПРЕПОДАВАТЕЛЬ

НП	ФИО	Стаж	КД
П1	Иванов И.М.	5	К1
П2	Петров М.И.	7	К2
П3	Сидоров Н.Г.	10	К3
П4	Егоров В.В.	5	К4

ДИСЦИПЛИНА(КД, Часы)

ДИСЦИПЛИНА

КД	Часы
К1	62
К2	74
К3	102
К4	80

Рис. 6.10. Отношения, полученные по правилу 2

В результате мы избежали пустых полей в отношениях, не потеряв данных. Добавив атрибут КД – ключ сущности ДИСЦИПЛИНА (с необязательным КП) в качестве внешнего ключа в отношение, соответствующее сущности ПРЕПОДАВАТЕЛЬ (с обязательным КП), мы связали отношения (рис. 6.11).

НП	ФИО	Стаж	КД

КД	Часы

Рис. 6.11. Связь отношений по внешнему ключу

Точнее говоря, мы создали условия для связывания отношений. Это связывание при работе с базой данных позволяет, например, получать одновременно данные о преподавателе и о ведущихся им дисциплинах (часах).

Правило 3. Если степень связи 1:1 и класс принадлежности обеих сущностей является необязательным, то необходимо использовать три отношения. Два отношения соответствуют связываемым сущностям, ключи которых являются первичными в этих отношениях. Третье отношение является связанным между первыми двумя, поэтому его ключ объединяет ключевые атрибуты связываемых отношений.

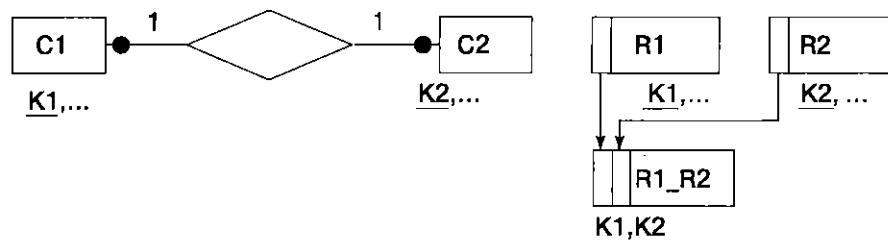


Рис. 6.12. Диаграмма и отношения для правила 3

На рис. 6.12 приведены диаграмма ER-типа и отношения, сформированные по правилу 3 на ее основе.

На рис. 6.13 приведены примеры отношений, подтверждающие необходимость использования трех отношений при наличии необязательного КП для обеих связываемых сущностей.

а) одно отношение

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

НП	ФИО	Стаж	КД	Часы
П1	Иванов И.М.	5	К1	62
П2	Петров М.И.	7	---	---
П3	Сидоров Н.Г.	10	К2	74
---	---	---	К3	102

б) два отношения

ПРЕПОДАВАТЕЛЬ

НП	ФИО	Стаж	КД
П1	Иванов И.М.	5	К1
П2	Петров М.И.	7	---
П3	Сидоров Н.Г.	10	К2

ДИСЦИПЛИНА

КД	Часы	НП
К1	62	П1
К2	74	П3
К3	102	---

в) три отношения

ПРЕПОДАВАТЕЛЬ

НП	ФИО	Стаж
П1	Иванов И.М.	5
П2	Петров М.И.	7
П3	Сидоров Н.Г.	10

ВЕДЕТ

НП	КД
П1	К1
П3	К2

ДИСЦИПЛИНА

КД	Часы
К1	62
К2	74
К3	102

Рис. 6.13. Варианты отношений для правила 3

Использование одного отношения в рассматриваемом случае приводит к наличию нежелательных пустых полей в этом отношении (рис. 6.13а). При использовании двух отношений (рис. 6.13б) нам пришлось добавить ключи каждой из сущностей в отношение, соответствующее другой сущности, чтобы не потерять сведения о том, какую дисциплину ведет каждый преподаватель и наоборот. При этом также появились пустые поля.

Выход заключается в использовании трех отношений, сформированных по правилу 3 (рис. 6.13в). Объектные отношения (с атрибутами сущностей) содержат полную информацию обо всех преподавателях и дисциплинах соответственно. Связное отношение ВЕДЕТ содержит данные о преподавателях, которые ведут дисциплины и о дисциплинах, которые ведутся преподавателями. При этом в нем имеется только одно упоминание о каждом преподавателе и дисциплине в силу связи 1:1. Это отношение содержит в данном случае только ключевые атрибуты обеих сущностей, но может иметь и другие атрибуты, характеризующие эту связь. Например, номер семестра, в котором преподаватель ведет дисциплину.

Итак, сформулированы три правила, позволяющие формировать отношения на основе ER-диаграмм, для вариантов со степенью связи типа 1:1. Сформулируем аналогичные два правила для вариантов, степень связи между сущностями которых 1:M.

Формирование отношений для связи 1:M

Если две сущности С1 и С2 связаны как 1:M, сущность С1 будем называть односвязной (1-связной), а сущность С2 – многосвязной (M-связной). Определяющим фактором при формировании отношений, связанных этим видом связи, является класс принадлежности M-связной сущности. Так, если класс принадлежности M-связной сущности обязательный, то в результате применения правила получим два отношения, если необязательный – три отношения. Класс принадлежности односвязной сущности не влияет на результат.

Чтобы убедиться в этом, рассмотрим отношение ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА (рис. 6.14), соответствующее диаграммам, приведенным на рис. 6.4, т. е. случаю, когда: связь типа 1:M, класс принадлежности M-связной сущности обязательный, 1-связной – необязательный.

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

НП	ФИО	Стаж	КД	Часы
П1	Иванов И.М.	5	К1	62
П1	Иванов И.М.	5	К2	74
П2	Петров М.И.	7	К4	80
П3	Сидоров Н.Г.	10	К5	96
П3	Сидоров Н.Г.	10	К6	120
П4	Егоров В.В.	5	К3	102
П4	Егоров В.В.	5	К7	89
П5	Козлов А.С.	8	---	---

Рис. 6.14. Исходное отношение

С отношением ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА (рис. 6.14) связанные следующие проблемы:

- имеются кортежи с пустыми полями (преподаватель не ведет дисциплины),
- избыточное дублирование данных (повторяется стаж преподавателя) в кортежах со сведениями о преподавателях, ведущих несколько дисциплин.

Если бы класс принадлежности 1-связной сущности был обязательным (нет преподавателя без дисциплины), то исчезли бы пустые поля, но повторяющиеся данные в атрибутах преподавателя сохранились бы. Для устранения названных проблем отношения могут быть сформированы по следующему правилу.

Правило 4. Если степень связи между сущностями 1:M (или M:1) и класс принадлежности M-связной сущности обязательный, то достаточно формирование двух отношений (по одному на каждую из сущностей). При этом первичными ключами этих отношений являются ключи их сущностей. Кроме того, ключ 1-связной сущности добавляется как атрибут (внешний ключ) в отношение, соответствующее M-связной сущности.

На рис. 6.15 приведены диаграмма ER-типа и отношения, сформированные по правилу 4.

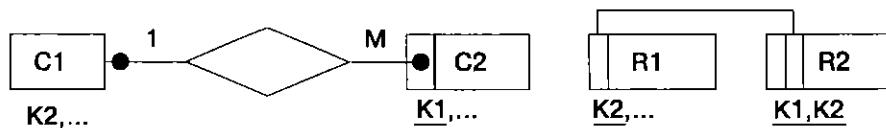


Рис. 6.15. Диаграмма и отношения для правила 4

В соответствии с правилом 4 преобразуем отношение на рис. 6.14 в два отношения (рис. 6.16).

ПРЕПОДАВАТЕЛЬ			ДИСЦИПЛИНА		
НП	ФИО	Стаж	КД	Часы	НП
П1	Иванов И.М.	5	K1	62	П1
П2	Петров М.И.	7	K2	74	П1
П3	Сидоров Н.Г.	10	K3	102	П4
П4	Егоров В.В.	5	K4	80	П2
П5	Козлов А.С.	8	K5	96	П3

Рис. 6.16. Отношения, полученные по правилу 4

Из рис. 6.16 видно, что пустые поля и дублирование информации удалось устраниТЬ. Потери сведений о том, кто из преподавателей ведет какую дисциплину, не произошло благодаря введению ключа НП сущности ПРЕПОДАВАТЕЛЬ в качестве внешнего ключа в отношение ДИСЦИПЛИНА.

Для формулирования и обоснования необходимости использования следующего правила рассмотрим следующий пример.

Пример. Связь между сущностями 1:М, а класс принадлежности М-связной сущности необязательный.

Пусть класс принадлежности 1-связной сущности также необязательный, хотя это и не принципиально, так как определяющим является класс принадлежности М-связной сущности. Посмотрим, к чему может привести использование одного отношения в этом случае (рис. 6.17).

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

НП	ФИО	Стаж	КД	Часы
П1	ИвановИ.М.	5	К1	62
П1	ИвановИ.М.	5	К2	74
П2	ПетровМ.И.	7	К4	80
---	---	---	К5	96
П3	Сидоров Н.Г.	10	К6	120
П4	Егоров В.В.	5	К3	102
П4	Егоров В.В.	5	К7	89
П5	Козлов А.С.	8	---	---

Рис. 6.17. Исходное отношение

С приведенным отношением связаны следующие проблемы:

1. Имеются пустые поля в кортежах, которые содержат следующее:

- а) данные о преподавателях, не ведущих дисциплин;
- б) данные о дисциплинах, которые не ведутся преподавателями.

2. Избыточное дублирование данных о преподавателях, ведущих более одной дисциплины.

В случае обязательного класса принадлежности 1-связной сущности исчезают проблемы 1 а). Для устранения всех проблем нужно перейти к трем отношениям в соответствии со следующим правилом.

Правило 5. Если степень связи 1:М (М:1) и класс принадлежности М-связной сущности является необязательным, то необходимо формирование трех отношений (рис. 6.18). Два отношения соответствуют связываемым сущностям, ключи которых являются первичными в этих отношениях. Третье отно-

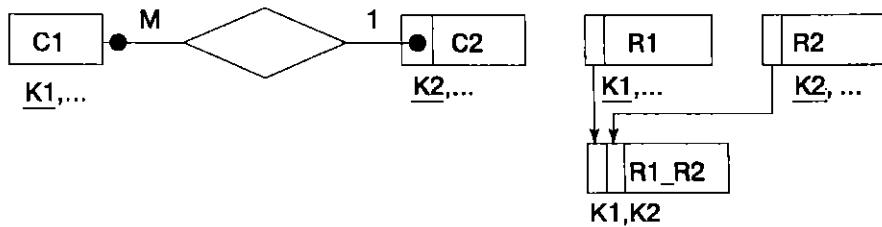


Рис. 6.18. Диаграмма и отношение для правила 5

шенис является связным между первыми двумя (его ключ объединяет ключевые атрибуты связываемых отношений).

В результате применения правила 5 к рассматриваемому отношению содержащиеся в нем данные (рис. 6.17) распределяются по трем отношениям (рис. 6.19).

ПРЕПОДАВАТЕЛЬ

НП	ФИО	Стаж
П1	Иванов И.М.	5
П2	Петров М.И.	7
П3	Сидоров Н.Г.	10
П4	Егоров В.В.	5
П5	Козлов А.С.	8

ВЕДЕТ

НП	КД
П1	К1
П1	К2
П2	К4
П3	К6
П4	К3
П4	К7

ДИСЦИПЛИНА

КД	Часы
К1	62
К2	74
К3	102
К4	80
К5	96
К6	120
К7	89

Рис. 6.19. Отношения, полученные по правилу 5

Таким образом, указанные проблемы удалось разрешить. Ключ в связном отношении ВЕДЕТ является составным и включает в себя ключевые атрибуты обоих связываемых отношений (сущностей). В практических ситуациях связное отношение может содержать и другие характеризующие связь атрибуты.

Подчеркнем, что определяющим фактором при выборе между 4-м или 5-м правилом является класс принадлежности M-связной сущности.

Формирование отношений для связи M:M

При наличии связи M:M между двумя сущностями необходимо три отношения независимо от класса принадлежности любой из сущностей. Исполь-

зование одного или двух отношений в этом случае не избавляет от пустых полей или избыточно дублируемых данных.

Правило 6. Если степень связи $M:M$, то независимо от класса принадлежности сущностей формируются три отношения. Два отношения соответствуют связываемым сущностям и их ключи являются первичными ключами этих отношений. Третье отношение является связным между первыми двумя, а его ключ объединяет ключевые атрибуты связываемых отношений.

На рис. 6.20 приведены диаграмма ER-типа и отношения, сформированные по правилу 6. Нами показан вариант с классом принадлежности сущностей $H-H$, хотя, согласно правилу 6, он может быть произвольным.

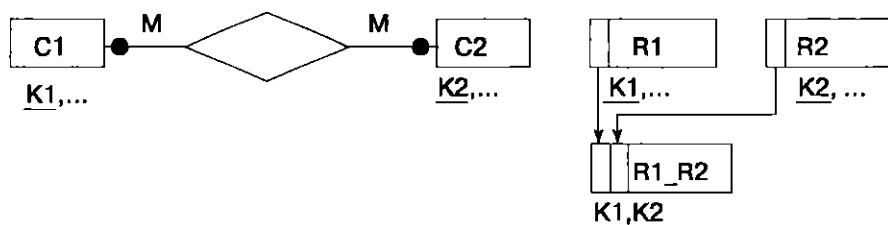


Рис. 6.20. Диаграмма и отношения для правила 6

Применим правило 6 к примеру, приведенному на рис. 6.5. В нем степень связи равна $M:M$, класс принадлежности для сущности ПРЕПОДАВАТЕЛЬ обязательный, а для сущности ДИСЦИПЛИНА – необязательный. Соответствующее этому примеру исходное отношение показано на рис. 6.21.

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

НП	ФИО	Стаж	КД	Часы
П1	ИвановИ.М.	5	K1	62
П1	ИвановИ.М.	5	K2	74
П2	ПетровМ.И.	7	K4	80
---	---	---	K3	102
П3	Сидоров Н.Г.	10	K6	120
П4	Егоров В.В.	5	K2	74
П4	Егоров В.В.	5	K7	89
П5	Козлов А.С.	8	K5	96

Рис. 6.21. Исходное отношение

В результате применения правила 6 получаются три отношения (рис. 6.22).

ПРЕПОДАВАТЕЛЬ			ВЕДЕТ		ДИСЦИПЛИНА	
НП	ФИО	Стаж	НП	КД	КД	Часы
П1	Иванов И.М.	5	П1	К1	К1	62
П2	Петров М.И.	7	П1	К2	К2	74
П3	Сидоров Н.Г.	10	П2	К4	К3	102
П4	Егоров В.В.	5	П3	К6	К4	80
П5	Козлов А.С.	8	П4	К3	К5	96
			П4	К7	К6	120
					К7	89

Рис. 6.22. Отношения, полученные по правилу 6

Аналогичные результаты получаются и для трех других вариантов, различающихся классами принадлежности их сущностей. Рассмотрим применение сформулированных правил на примерах проектирования БД методом сущность-связь.

6.4. Пример проектирования БД учебной части

Применим описанные правила к примеру, рассмотренному при изучении метода нормальных форм (подраздел 5.2). Напомним, что речь шла о БД для учебной части, содержащей следующие сведения:

ФИО – фамилия и инициалы преподавателя (возможность совпадения фамилии и инициалов у преподавателей исключена).

Должн – должность, занимаемая преподавателем.

Оклад – оклад преподавателя.

Стаж – преподавательский стаж.

Д_Стаж – надбавка за стаж.

Каф – номер кафедры, на которой работает преподаватель.

Предм – название дисциплины (предмета), ведомой преподавателем.

Группа – номер группы, в которой преподаватель проводит занятия.

ВидЗан – вид занятий, проводимых преподавателем в учебной группе (преподаватель в одной группе ведет только один вид занятий).

Исходное отношение ПРЕПОДАВАТЕЛЬ приведено на рис. 5.4.

При проектировании будем придерживаться этапов, описанных в подразделе 6.2.

Первый этап проектирования – выделение сущностей и связей между ними.

Выделим следующие сущности:

- ПРЕПОДАВАТЕЛЬ (Ключ – ФИО),
- ЗАНЯТИЕ (Ключ – Группа, Предм),
- СТАЖ (Ключ – Стаж),
- ДОЛЖНОСТЬ (Ключ – Должн).

Выделим связи между сущностями:

- ПРЕПОДАВАТЕЛЬ **ИМЕЕТ** СТАЖ,
- ПРЕПОДАВАТЕЛЬ **ВЕДЕТ** ЗАНЯТИЕ,
- ПРЕПОДАВАТЕЛЬ **ЗАНИМАЕТ** ДОЛЖНОСТЬ.

Второй этап проектирования – построение диаграммы ER-типа с учетом всех сущностей и связей между ними. Диаграмма ER-типа для рассматриваемого примера приведена на рис. 6.23.

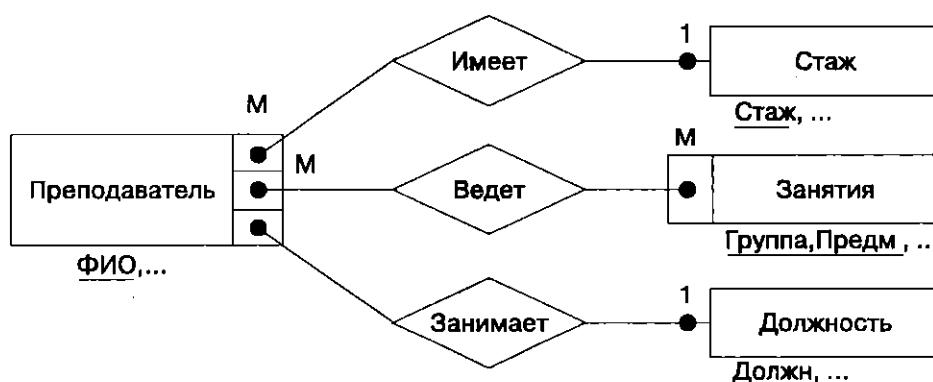


Рис. 6.23. Диаграмма ER-типа

Связь **ИМЕЕТ** является связью типа M:1, т. к. одинаковый стаж могут иметь несколько преподавателей. Сущность ПРЕПОДАВАТЕЛЬ имеет обязательный класс принадлежности, поскольку каждый преподаватель имеет свой стаж. Сущность СТАЖ имеет необязательный класс принадлежности, так как возможны такие значения стажа, которые не имеет ни один из преподавателей.

Связь **ВЕДЕТ** имеет тип M:M, так как преподаватель может вести несколько занятий, а каждое занятие может проводиться несколькими преподавателями. Занятие может быть лекционным или практическим, проводимым преподавателем в учебной группе по одной из дисциплин. Обе сущности в данной связи имеют КП обязательный, в предположении, что нет преподавателей, которые не проводят занятий, и нет занятий, которые не обеспечены преподавателями.

Связь ЗАНИМАЕТ имеет тип М:1, так как каждый преподаватель занимает определенную должность и одинаковые должности могут занимать несколько преподавателей. Сущность ПРЕПОДАВАТЕЛЬ имеет обязательный класс принадлежности, так как предполагаем, что каждый преподаватель занимает должность. Сущность ДОЛЖНОСТЬ имеет необязательный КП, так как не исключаем, например, отсутствие должности профессора на кафедре, а значит, и преподавателя, который ее занимает.

Третий этап проектирования – формирование набора предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения, используя диаграммы ER-типа.

На основе анализа диаграммы ER-типа (рис. 6.23) с помощью шести сформулированных выше правил получаем набор предварительных отношений, представленных следующими схемами отношений.

Связь ИМЕЕТ удовлетворяет условиям правила 4, в соответствии с которым получаем два отношения:

1. ПРЕПОДАВАТЕЛЬ (ФИО, Стаж, ...)
2. СТАЖ (Стаж, ...).

Связь ВЕДЕТ удовлетворяет условиям правила 6, в соответствии с которым получаем три отношения:

1. ПРЕПОДАВАТЕЛЬ (ФИО, Стаж, ...).
2. ЗАНЯТИЕ (Группа, Предм., ...).
3. ВЕДЕТ (ФИО, Группа, Предм., ...).

Связь ЗАНИМАЕТ аналогично связи ИМЕЕТ удовлетворяет условиям правила 4, поэтому имеем следующие два отношения

1. ПРЕПОДАВАТЕЛЬ (ФИО, Стаж, Должн., ...)
2. ДОЛЖНОСТЬ (Должн., ...).

Третий этап проектирования – добавление неключевых атрибутов, которые не были выбраны в качестве ключевых раньше, и назначение их одному из предварительных отношений с тем условием, чтобы отношения отвечали требованиям нормальной формы Бойса – Кодда.

После добавления неключевых атрибутов схемы отношений примут следующий вид:

- ПРЕПОДАВАТЕЛЬ (ФИО, Стаж, Должн., Каф),
 СТАЖ (Стаж, Д_Стаж),
 ЗАНЯТИЕ (Группа, Предм.),
 ВЕДЕТ (ФИО, Группа, Предм., ВидЗан),
 ДОЛЖНОСТЬ (Должн., Оклад).

После определения отношений следует проверить их на соответствие требованиям нормальной формы Бойса – Кодда. В нашем случае мы получили те же отношения, что и при проектировании примера методом нормальных форм. Полученная схема базы данных приведена на рис. 6.24.

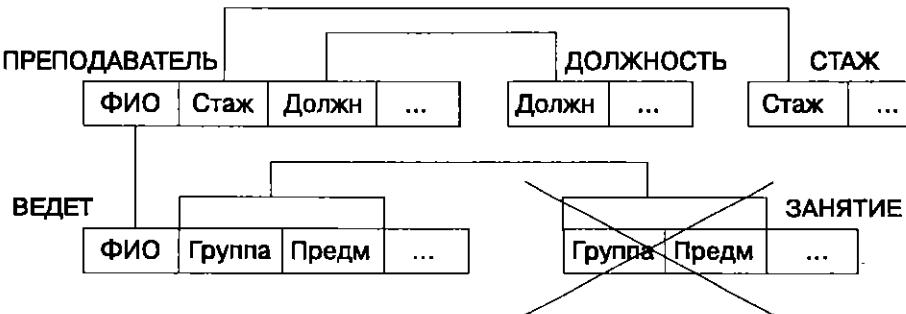


Рис. 6.24. Схема базы данных

Замечания.

- В нашем примере отношение ЗАНЯТИЕ, кроме ключевых атрибутов (Группа, Предм), других атрибутов не имеет. Поэтому оно не несет дополнительной информации, кроме содержащейся в отношении ВЕДЕТ. Действительно, это отношение включает в себя атрибуты Группа, Предм. Поэтому отношение ЗАНЯТИЕ нужно исключить из формируемой схемы БД (на рис. 6.24 оно перечеркнуто). Если бы имелись другие атрибуты, например атрибут Семестр, в котором некоторая группа изучает конкретную дисциплину, то получили бы отношение ЗАНЯТИЕ (Группа, Предм, Семестр), которое вошло бы в БД.
- На последнем этапе проектирования предварительные отношения анализируются на предмет избыточного дублирования информации. При этом возможно рассмотрение нескольких кортежей каждого отношения. При наличии избыточности возможно либо перепроектирование соответствующей части проекта (ER-диаграмм), либо декомпозиция соответствующих отношений с использованием метода нормальных форм. Конечный результат преобразований представляет собой совокупность отношений в нормальной форме Бойса – Кодда.
- Рассмотренные правила проектирования БД позволяют моделировать многие практические ситуации. Построение ряда других реальных моделей может потребовать использования дополнительных конструкций. В частности, может возникнуть необходимость использования связей более высокого порядка, чем *бинарные*, например, *тернарные*, связывающие три сущности.

Контрольные вопросы и задания

1. Перечислите основные понятия метода сущность-связь.
2. Охарактеризуйте понятие ключа сущности.

3. Что представляют собой диаграммы ER-экземпляров и диаграммы ER-типа?
4. Что определяет степень связи между сущностями?
5. Каким может быть класс принадлежности?
6. Приведите пример диаграммы ER-экземпляров со степенью связи между сущностями 1:1 и обязательным классом принадлежности двух сущностей.
7. Как на диаграммах ER-типа обозначаются степень связи, обязательное и необязательное участие в связи экземпляров сущности?
8. Приведите пример диаграммы ER-экземпляров для связи типа 1:M варианта Н-О.
9. Назовите этапы проектирования базы данных.
10. Как осуществляется формирование отношений для связи 1:1?
11. Сформулируйте правило 3 формирования отношений, если степень связи 1:1 и класс принадлежности обеих сущностей является необязательным.
12. Сформулируйте правило формирования отношения для случая степени связи между сущностями 1:M (M:1) и обязательного класса принадлежности M-связной сущности.
13. Укажите правила формирования отношений для связи M:M.
14. Покажите, что полученные в прикладном примере из раздела отношения находятся в нормальной форме Бойса – Кодда.

Литература

1. Дейт К. Дж. Введение в системы баз данных / Пер. с англ. – 6-е изд. К.: Диалектика, 1998.
2. Джексон Г. Проектирование реляционных баз данных для использования с МИКРОЭВМ / Пер. с англ. М.: Мир, 1991.
3. Кузнецов С. Д. Введение в СУБД: часть 4 // Системы Управления Базами Данных, № 4, 1995. С. 114–122.
4. Основы современных компьютерных технологий: Учебник / Под ред. проф. Хомоненко А. Д. Авторы: Брякалов Г. А., Войцеховский С. В., Воробьев Е. и др. – СПб.: КОРОНА прнт, 2005. – 672 с.
5. Хансен Г., Хансен Д. Базы данных: разработка и управление / Пер. с англ. М.: ЗАО «Издательство БИНОМ», 1999.

7. Средства автоматизации проектирования

В разделе дается общая характеристика CASE-средств создания и сопровождения информационных систем. Рассматриваются модели жизненного цикла программных систем, описываются модели структурного и объектно-ориентированного проектирования. Дается классификация CASE-средств и приводится характеристика наиболее популярных систем.

7.1. Основные определения

Для автоматизации проектирования и разработки информационных систем в 70–80-е гг. широко применялась *структурная* методология, означающая использование формализованных методов описания разрабатываемой системы и принимаемых технических решений. При этом использовались графические средства описания различных моделей информационных систем с помощью схем и диаграмм. При ручной разработке информационных систем такие графические модели разрабатывать и использовать весьма трудоемко.

Отмеченные обстоятельства послужили одной из причин появления программно-технологических средств, получивших название CASE-средств и реализующих CASE-технологию создания и сопровождения информационных систем. Кроме структурной методологии, в ряде современных CASE-средств используется объектно-ориентированная методология проектирования.

Термин CASE (Computer Aided Software Engineering) дословно переводится как разработка программного обеспечения с помощью компьютера. В настящее время этот термин получил более широкий смысл, означающий автоматизацию разработки информационных систем.

CASE-средства представляют собой программные средства, поддерживающие процессы создания и/или сопровождения информационных систем, такие как: анализ и формулировка требований, проектирование баз данных и приложений, генерация кода, тестирование, обеспечение качества, управление конфигурацией и проектом.

CASE-систему можно определить как набор CASE-средств, имеющих определенное функциональное назначение и выполненных в рамках единого программного продукта.

CASE-технология обычно определяется как методология проектирования информационных систем плюс инструментальные средства, позволяющие наглядно моделировать предметную область, анализировать ее модель на всех этапах разработки и сопровождения информационной системы и разрабатывать приложения для пользователей.

CASE-индустрия объединяет сотни фирм и компаний различной ориентации. Практически все серьезные зарубежные программные проекты осуществляются с использованием CASE-средств, а общее число распространяемых пакетов превышает 500 наименований.

Основная цель CASE-систем и средств состоит в том, чтобы отделить проектирование программного обеспечения от его кодирования и последующих этапов разработки (тестирование, документирование и пр.), а также автоматизировать весь процесс создания программных систем, или *инжиниринг* (от англ. engineering – разработка).

В последнее время все чаще разработка программ начинается с некоторого предварительного варианта системы. В качестве такого варианта может выступать специально для этого разработанный *прототип*, либо устаревшая и не удовлетворяющая новым требованиям система. В последнем случае для восстановления знаний о программной системе с целью последующего их использования применяют повторную разработку – *ренинжиниринг*.

Повторная разработка сводится к построению исходной модели программной системы путем исследования ее программных кодов. Имея модель, можно ее усовершенствовать, а затем вновь перейти к разработке. Так часто и поступают при проектировании. Одним из наиболее известных принципов такого типа является принцип «возвратного проектирования» – Round Trip Engineering (RTE).

Современные CASE-системы не ограничиваются только разработкой, а чаще всего обеспечивают и повторную разработку. Это существенно ускоряет разработку приложений и повышает их качество.

Перспективная CASE-система

Динамика изменения позиций структурных и объектно-ориентированных систем позволяет предположить, что перспективная CASE-система будет объектно-ориентированной. Рассмотрим требования к идеальной перспективной CASE-системе.

Полнофункциональная объектно-ориентированная система должна решать задачи анализа и моделирования, проектирования, разработки (реализации), а также иметь эффективную инфраструктуру, обеспечивающую сервисом первые три основные задачи.

Для решения задач анализа и моделирования целесообразно иметь интегрированную среду разработчика, которая должна обеспечивать возможности:

- динамического моделирования событий в системе;

- динамической и согласованной коррекции всей совокупности диаграмм;
- добавления пояснительных надписей к диаграммам моделей и в документацию;
- автоматической генерации документации;
- создания различных представлений и скрытия неиспользуемых слоев системы;
- поддержки различных нотаций (это требование ослабевает в связи с переходом к единому языку моделирования).

Осуществление процесса *проектирования* предполагает наличие возможностей:

- определения основной модели прикладной задачи (бизнес-модели, обычно объектно-ориентированной) и правил ее поведения (бизнес-правил);
- поддержки процесса проектирования с помощью библиотек, оснащенных средствами хранения, поиска и выбора элементов проектирования (объектов и правил);
- создания пользовательского интерфейса и поддержания распределенных программных интерфейсов (поддержка стандартов OLE, OpenDoc, доступ к библиотекам HTML/Java и т. п.);
- создания различных распределенных клиент-серверных приложений.

Средства разработки в составе CASE-системы должны обеспечивать построение приложения по результатам предыдущих этапов разработки приложения. Максимально высоким требованием к средствам разработки можно считать генерацию готовой к выполнению программы.

CASE-системы *ближайшего будущего* должны позволять создавать скелетные заготовки под классы, атрибуты и методы, готовые приложения на объектно-ориентированных языках программирования типа C++ или Smalltalk, либо программы на языках клиент-серверных продуктов (PowerBuilder, Forte, VisualAge, VisualWorks и т. д.). К поддерживаемым языкам, по-видимому, скоро добавится язык Java.

Для обеспечения связи с другими этапами жизненного цикла приложения средства CASE-системы должны включать и функции контроля программного кода на синтаксическую корректность и соответствие моделям.

Ядром инфраструктуры будущих CASE-систем должен быть репозиторий, отвечающий за генерацию кода, реинжиниринг и обеспечивающий соответствие между моделями и программными кодами. Основу репозитория составят объектно-ориентированные БД, хотя ранее для этого использовались реляционные БД. Другими важнейшими функциями инфраструктуры являются функции контроля версий и управления частями системы при коллективной разработке.

7.2. Модели жизненного цикла

Модель жизненного цикла (ЖЦ) цикла программного обеспечения информационной системы (ПО ИС) при автоматизированном проектировании ИС играет достаточно важную роль. Это обусловлено тем, что каждая из CASE-систем ориентирована на (поддерживает) определенную модель ЖЦ ПО ИС.

Жизненный цикл ПО ИС представляет собой непрерывный процесс, начинающийся с момента принятия решения о создании ПО и заканчивающийся при завершении его эксплуатации.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 (International Organization of Standardization – Международная организация по стандартизации/International Electrotechnical Commission – международная комиссия по электротехнике). В нем определяется структура ЖЦ, содержащая процессы, действия и задачи, которые должны быть выполнены при создании ПО.

Под **моделью ЖЦ** ПО понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении ЖЦ. Наибольшее распространение получили следующие модели ЖЦ ПО: каскадная, с промежуточным контролем и спиральная.

Модели каскадная и с промежуточным контролем включают следующие этапы жизненного цикла ПО: анализ, проектирование, реализация, внедрение и сопровождение.

Каскадная модель предполагает строго последовательную реализацию перечисленных этапов жизненного цикла. *Достоинствами* такой модели являются: формирование на каждом этапе законченного комплекта документации и возможность планирования сроков завершения работ и соответствующих затрат. *Недостатком* модели является ее несоответствие реальному процессу создания ПО, который обычно не укладывается в жесткую схему и требует возврата к предыдущим этапам для уточнения или пересмотра принятых решений.

Модель с промежуточным контролем приближает жизненный цикл к реальному процессу создания и применения ПО. В отличие от каскадной модели, она допускает возврат с каждого этапа жизненного цикла на любой предыдущий этап для выполнения межэтапной корректировки. При этом обеспечивается большая надежность ПО, но вместе с тем увеличивается длительность периода разработки.

Спиральная модель жизненного цикла (рис. 7.1) позволяет устранить недостатки предыдущих моделей. Основной упор в ней делается на начальные этапы: анализ и проектирование. На них реализуемость технических решений проверяется с помощью создания прототипов.

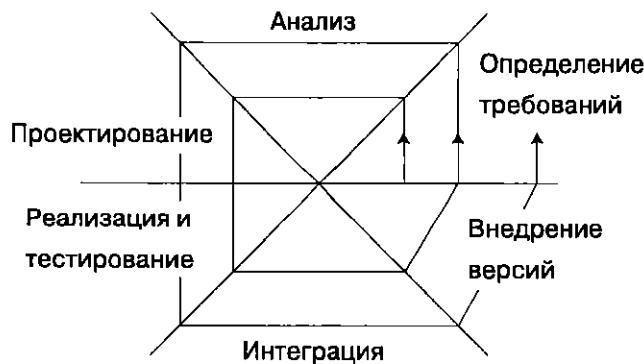


Рис. 7.1. Спиральная модель жизненного цикла

При спиральной схеме разработки неполное завершение работ на очередном этапе позволяет переходить на следующий этап. Незавершенная работа может выполняться на следующем витке спирали. Тем самым обеспечивается возможность предъявить пользователям системы ее некоторый работоспособный вариант для уточнения требований.

7.3. Модели структурного проектирования

Структурный подход к анализу и проектированию информационной системы заключается в рассмотрении ее с общих позиций с последующей детализацией и представлением в виде иерархической структуры. На верхнем уровне иерархии обычно представляется функциональное описание системы.

При проведении структурного анализа и проектирования для повышения наглядности используется графическое представление функций информационной системы и отношений между данными.

Наиболее распространенными моделями и диаграммами графического представления являются следующие:

- диаграммы сущность-связь или ER-диаграммы – Entity-Relationship Diagrams (ERD) служат для наглядного представления схем баз данных (раздел 6);
- диаграммы потоков данных – Data Flow Diagrams (DFD) служат для иерархического описания модели системы;
- метод структурного анализа и проектирования – Structured Analysis and Design Technique (SADT), служащий для построения функциональной модели объекта;
- схемы описания иерархии вход-обработка-выход – Hierarchy plus Input-Processing-Output (HIPO) служат для описания реализуемых программой функций и циркулирующих внутри нее потоков данных;

- диаграммы Варнье-Орра служат для описания иерархической структуры системы с выделением элементарных составных частей, выделением процессов и указанием потоков данных для каждого процесса.

Названные модели позволяют получить описание информационной системы, а их состав зависит от требуемой полноты ее описания. Широко используемые диаграммы сущность-связь описаны в разделе 6. Рассмотрим кратко также важные и часто используемые в CASE-средствах диаграммы и модели DFD и SADT.

Диаграммы потоков данных

Диаграммы потоков данных DFD лежат в основе методологии моделирования потоков данных, при котором модель системы строится как иерархия диаграмм потоков данных, описывающих процесс преобразования от ее входа до выдачи пользователю.

Диаграммы верхних уровней иерархии, или контекстные диаграммы, определяют основные процессы или подсистемы информационной системы с внешними входами и выходами. Их декомпозиция выполняется с помощью диаграмм более низкого уровня, вплоть до элементарных процессов.

Основными компонентами диаграмм потоков данных являются:

- *внешние сущности* – источники или потребители информации, порождающие или принимающие информационные потоки (потоки данных);
- *системы/подсистемы*, преобразующие получаемую информацию и порождающие новые потоки;
- *процессы*, представляющие преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом;
- *накопители данных*, представляющие собой абстрактное устройство для хранения информации, которую можно поместить в накопитель и через некоторое время извлечь;
- *потоки данных*, определяющие информацию, передаваемую через некоторое соединение от источника к приемнику.

Опишем типовой набор графических блоков, который обычно используют для обозначения компонентов DFD. В конкретных CASE-средствах и системах этот набор может иметь некоторые отличия.

Внешняя сущность обозначается прямоугольником с тенью. *Система и подсистема* изображаются в форме прямоугольника с полями: номер, имя с определениями и дополнениями и имя проектировщика. *Процесс* изображается в форме прямоугольника с полями: номер, имя (содержит наименование процесса в виде предложения сделать то-то) и физической реализации (указывает, какое подразделение, программа или устройство выполняет процесс). *Накопитель данных* изображается в форме прямоугольника без правой (или правой и левой) линии границы: идентификатор (буква D с

числом) и имя (указывает на хранимые данные). *Поток данных* изображается линией со стрелкой, показывающей направление потока, и именем, отражающим его содержание.

Примеры фрагментов диаграммы потоков данных с изображением перечисленных компонентов приведены на рис. 7.2.

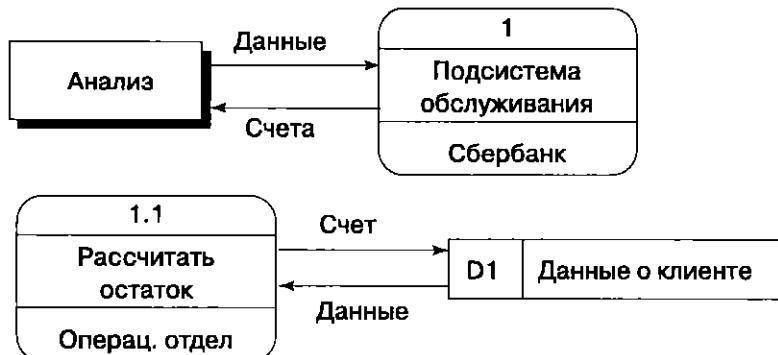


Рис. 7.2. Фрагменты диаграммы потоков данных

Построение иерархии диаграмм потоков данных начинается с построения контекстной диаграммы. В случае простой информационной системы можно ограничиться одной контекстной диаграммой. Применительно к сложной информационной системе требуется построение иерархии контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит набор подсистем, соединенных потоками данных.

Методология функционального моделирования

Методология функционального моделирования SADT служит для построения функциональной модели объекта какой-либо предметной области. Последняя отображает функциональную структуру объекта – выполняемые им действия и связи между ними.

Функциональная модель информационной системы состоит из имеющих ссылки друг к другу диаграмм, фрагментов текстов и глоссария. На диаграммах представляются функции ИС и взаимосвязи (интерфейсы) между ними в виде блоков и дуг. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация указывается сверху, обрабатываемая информация – с левой стороны блока, выводимая информация – с правой стороны, выполняющий операцию механизм (человек, программа или устройство), представляется дугой снизу блока (см. рис. 7.3).

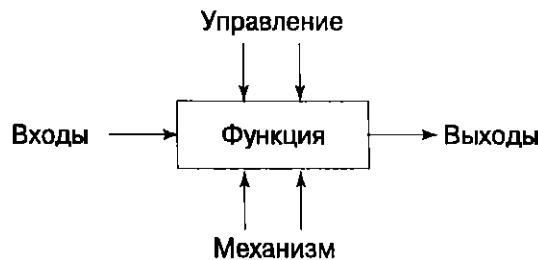


Рис. 7.3. Функциональный блок и дуги интерфейса

При использовании методологии SADT выполняется постепенное наращивание степени детализации в построении модели информационной системы. На рис. 7.4 показана декомпозиция исходного блока системы на три составляющих компонента. Каждый из блоков определяет подфункции исходной функции и, в свою очередь, может быть декомпозирован аналогичным образом для обеспечения большей детализации.

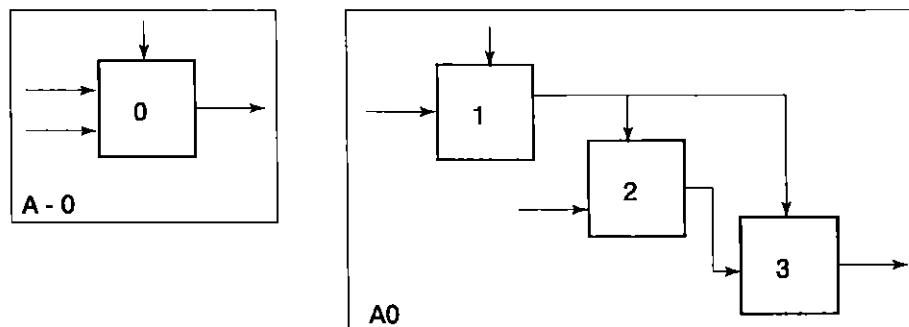


Рис. 7.4. Декомпозиция диаграмм

В общем случае функциональная модель ИС представляет собой серию диаграмм с документацией, декомпозирующих сложный объект на составные компоненты в виде блоков. Блоки на диаграмме нумеруются. Для указания положения диаграммы или блока в иерархии диаграмм используются номера диаграмм. Например, обозначение А32 указывает на диаграмму, детализирующую блок 2 на диаграмме А3. В свою очередь, диаграмма А03 детализирует блок 3 на диаграмме А0.

На диаграммах функциональной модели SADT последовательность и время явно не указываются. Обратные связи, итерации, процессы и перекрывающиеся по времени функции можно отобразить с помощью дуг.

В методологии функционального моделирования существенным свойством является отображение возможных типов связей между функциями. Можно выделить следующие семь типов связей в порядке возрастания значимости:

- *случайные связи*, означающие, что связь между функциями мала или отсутствует;
- *логические связи*, означающие, что данные и функции относятся к одному классу или набору элементов, но функциональных отношений между ними нет;
- *временные связи* представляют функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно;
- *процедурные связи* означают, что функции группируются вместе, так как выполняются в течение одной и той же части цикла или процесса;
- *коммуникационные связи* означают, что функции группируются вместе, так как используют одни и те же входные данные и/или порождают одни и те же выходные данные;
- *последовательные связи* служат для обозначения причинно-следственной зависимости – выходные данные одной функции являются входными данными другой функции;
- *функциональные связи* обозначают случай, когда все элементы функции влияют на выполнение только одной функции.

7.4. Объектно-ориентированные модели

Большинство моделей объектно-ориентированного проектирования близки по возможностям, но имеют отличия в основном в форме представления. Популярность объектно-ориентированных технологий привела к сближению большинства известных моделей. Многообразие моделей порождает трудности проектировщиков по выбору модели и по обмену информацией при работе над разными проектами. В этой связи известные специалисты Г.Буч, Д.Рамбо и И.Джекобсон при поддержке фирмы Rational Software Corporation провели работу над унифицированной моделью и методом, получившим название UML (Unified Modeling Language – унифицированный язык моделирования).

Общая характеристика UML

UML представляет собой единый язык моделирования, предназначенный для спецификации, визуализации, конструирования и документирования материалов программных систем, а также для моделирования бизнеса и дру-

гих непрограммных систем. В снову создания UML положены три наиболее распространенные модели:

- Booch, получившая название по фамилии автора Гради Буча (Grady Booch);
- OMT (Object Modeling Technique – метод моделирования объектов);
- OOSE (Object-Oriented Software Engineering – объектно-ориентированное проектирование программного обеспечения).

На заключительной стадии разработки, унификации и принятия UML в качестве стандарта большой вклад внес консорциум OMG (Object Management Group – группа управления объектом). UML можно определить также как промышленный объектно-ориентированный *стандарт моделирования*. Он включает в себя в унифицированном виде лучшие методы визуального (графического) моделирования. В настоящее время имеется целый ряд инструментальных средств, производители которых заявляют о поддержке UML, среди них можно выделить: Rational Rose, Select Enterprise, Platinum и Visual Modeler.

Типы диаграмм UML

Создаваемый с помощью UML проект информационной системы может включать в себя следующие 8 видов диаграмм (diagrams):

- прецедентов использования (use case),
- классов (class),
- состояний (statechart),
- активности (activity),
- следования (sequence),
- сотрудничества (collaboration),
- компонентов (component),
- размещения (deployment).

Диаграммы состояний, активности, следования и сотрудничества образуют набор диаграмм, служащих для описания *поведения* разрабатываемой информационной системы. Причем, последние две обеспечивают описание *взаимодействия объектов* информационной системы. Диаграммы компонентов и размещения описывают физическую реализацию информационной системы.

Каждая из диаграмм может содержать элементы определенного типа. Типы допустимых элементов и отношений между ними зависят от вида диаграммы. Охарактеризуем указанные виды диаграмм более подробно.

Диаграммы *прецедентов использования* описывают функциональность ИС, видимую пользователями системы. Каждая функциональность изображается в виде прецедентов использования. Прецедент – это ти-

личное взаимодействие пользователя с системой, которое выполняет следующее:

- описывает видимую пользователем функцию,
- представляет различные уровни детализации,
- обеспечивает достижение конкретной цели.

Прецедент изображается как овал, связанный с типичными пользователями, называемыми «актерами» (actors). Актером является любая сущность, взаимодействующая с системой извне, например человек, оборудование, другая система. Прецедент описывает, что система предоставляет актеру – определяет набор транзакций, выполняемых актером при диалоге с информационной системой. На диаграмме изображается один актер, но пользователей, выступающих в роли актера, может быть много. Диаграмма прецедентов использования имеет высокий уровень абстракции и позволяет определить функциональные требования к ИС.

Диаграммы **классов** описывают *статическую* структуру классов. В состав диаграмм классов входят следующие элементы: классы, объекты и отношения между ними. Класс представляется прямоугольником, включающим три раздела: имя класса, атрибуты и операции. Аналогичное обозначение применяется и для объектов, с той разницей, что к имени класса добавляется имя объекта и вся надпись подчеркивается.

Допускается отображение дополнительной информации (абстрактные операции и классы, стереотипы, общие и частные методы, интерфейсы, и т. д.). Ассоциации, или статические связи между классами, изображаются в виде связующей линии, на которой может указываться мощность ассоциации, направление, название, и возможное ограничение. Можно отразить свойства ассоциации, например отношение агрегации, когда составными частями класса являются другие классы. Отношение агрегации изображается в виде ромба, расположенного рядом с агрегирующим классом. Отношение обобщения изображается в виде треугольника и связующей линии, позволяя представить иерархию наследования классов.

Диаграммы **состояний** описывают поведение объекта во времени, моделируют все возможные изменения в состоянии объекта, вызванные внешними воздействиями со стороны других объектов или извне. Диаграммы состояний применяются для описания поведения объектов и для описания операций классов. Этот тип диаграмм описывает изменение состояния одного класса или объекта. Каждое состояние объекта представляется в виде прямоугольника с закругленными углами, содержащего имя состояния и, возможно, значение атрибутов объекта в данный момент времени. Переход осуществляется при наступлении некоторого события (например, получения объектом сообщения или приема сигнала) и изображается в виде стрелки, соединяющей два соседних состояния. Имя события указывается на переходе. На переходе могут указываться также действия, производимые объектом в ответ на внешние события.

Диаграммы **активности** представляют частный случай диаграмм состояний. Каждое состояние есть выполнение некоторой операции, и переход в следующее состояние происходит при завершении операции в исходном состоянии. Тем самым реализуется процедурное, синхронное управление, обусловленное завершением внутренних действий. Описываемое состояние не имеет внутренних переходов и переходов по внешним событиям.

Для диаграмм активности используются аналогичные диаграммам состояний обозначения, но на переходах отсутствует сигнатура события и добавлен символ «синхронизации» переходов для реализации параллельных алгоритмов. Диаграммы активности используются в основном для описания операций классов.

Диаграмма **следования** определяет временную последовательность (динамику взаимодействия) передаваемых сообщений, порядок, вид и имя сообщения. На диаграмме изображаются объекты, непосредственно участвующие во взаимодействии, и не показываются статические ассоциации с другими объектами.

Диаграмма **следования** имеет два измерения. Первое – слева направо, в виде вертикальных линий, изображающих объекты, участвующие во взаимодействии. Верхняя часть линий дополняется прямоугольником, содержащим имя класса объекта или имя экземпляра объекта. Второе измерение – вертикальная временная ось. Посылаемые сообщения изображаются в виде стрелок с именем сообщения и упорядочены по времени возникновения.

Диаграммы **сотрудничества** описывают взаимодействие объектов системы, выполняемого ими для получения некоторого результата. Под получением результата подразумевается выполнение законченного действия, например, описание в терминах взаимодействующих объектов смоделированного ранее *прецедента использования* информационной системы или некоторого сервиса, объявленного как операция класса на соответствующей диаграмме.

Диаграмма сотрудничества изображает объекты, участвующие во взаимодействии, в виде прямоугольников, содержащих имя объекта, его класс и, возможно, значение атрибутов. Ассоциации между объектами изображаются в виде соединительных линий. Возможно указание имени ассоциации и ролей объектов в данной ассоциации. Динамические связи (потоки сообщений) представляются в виде соединительных линий между объектами, сверху которых располагается стрелка с указанием направления и имени сообщения.

Диаграмма **компонентов** служит для определения архитектуры разрабатываемой системы путем установления зависимости между программными компонентами: исходным, бинарным и/или исполняемым кодом. Во многих средах разработки модуль, или компонент, соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости

(как при компиляции).

Диаграммы ***размещения*** используются для задания конфигурации компонентов, процессов и объектов, действующих в системе на этапе выполнения. Кроме того, они показывают физическую зависимость аппаратных устройств, участвующих в реализации системы, и соединений между ними – маршрутов передачи информации.

Примеры диаграмм UML

Чтобы получить более наглядное представление, приведем ряд диаграмм UML. Рассмотрим пример, в котором описана объектная модель, построенная в Rational Rose 98. В качестве предметной области используем описание работы библиотеки, которая получает запросы от клиентов на различные издания и регистрирует информацию об их возвращении в фонды библиотеки.

Пример диаграммы *предшественников использования* приведен на рис. 7.5. На диаграмме приведен ряд выделенных при анализе реализуемых информационной системой функций: администрирование пользователей (Administrative Client); учет книг (Administrative Books); составление отчетов (Report) и поиск издания (Find Book).

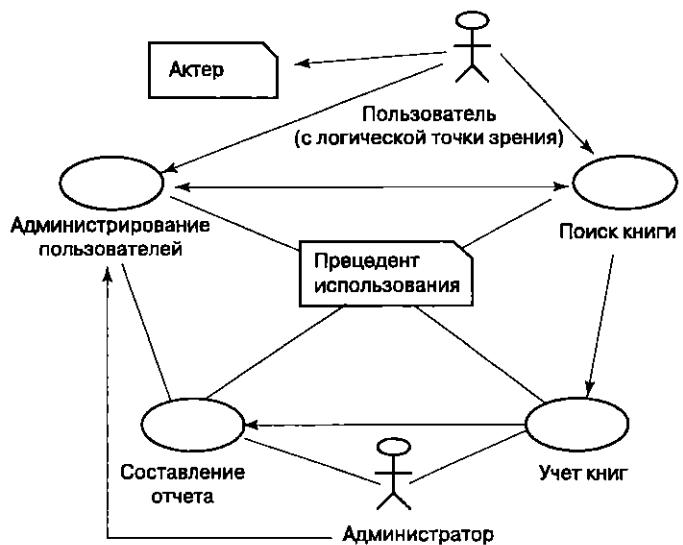


Рис. 7.5. Диаграмма прецедентов использования

Пример диаграммы следования приведен на рис. 7.6. Приведенная диаграмма описывает поведение объектов во времени. Она показывает объекты и последовательность сообщений, посылаемых объектами.



Рис. 7.6. Диаграмма следования

Отметим, что построение модели ИС до ее программной разработки является необходимым этапом проектирования. Хорошие модели позволяют наладить конструктивное взаимодействие между заказчиками, пользователями и разработчиками. Диаграммы UML обеспечивают ясное представление архитектурных решений для разрабатываемой системы. Сложность информационных систем растет и как следствие возрастает актуальность применения эффективных языков моделирования, таких как UML.

7.5. Классификация CASE-средств

При классификации CASE-средств используют следующие признаки:

- ориентацию на этапы жизненного цикла;
- функциональную полноту;
- тип используемой модели;
- степень независимости от СУБД;
- допустимые платформы.

Рассмотрим классификацию CASE-средств по наиболее часто используемым признакам.

По *ориентации* на этапы жизненного цикла выделяют следующие основные типы CASE-средств:

- *средства анализа*, предназначенные для построения и анализа моделей предметной области, например: Design/IDEF (Meta Software) и BPwin (Logic Works);

- *средства анализа и проектирования*, обеспечивающие создание проектных спецификаций, например: Vantage Team Builder (Cayenne), Silverrun (Silverrun Technologies), PRO-IV (McDonnell Douglas) и CASE.Аналитик (МакроПроджект);
- *средства проектирования баз данных*, обеспечивающие моделирование данных и разработку схем баз данных для основных СУБД, например: ERwin (Logic Works), S-Designor (SPD), DataBase Designer (ORACLE);
- *средства разработки приложений*, например: Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Centura) и Delphi (Borland).

По *функциональной полноте* CASE-системы и средства можно условно разделить на следующие типы:

- системы, предназначенные для решения частных задач на одном или нескольких этапах жизненного цикла, например, ERwin (Logic Works), S-Designor (SPD), CASE.Аналитик (МакроПроджект) и Silverrun (Silverrun Technologies);
- *интегрированные системы*, поддерживающие весь жизненный цикл ИС и связанные с общим репозиторием, например система Vantage Team Builder (Cayenne) и система Designer/2000 с системой разработки приложений Developer/2000 (ORACLE).

По *типу используемых моделей* CASE-системы условно можно разделить на три основные разновидности: структурные, объектно-ориентированные и комбинированные.

Исторически первые *структурные* CASE-системы основаны на методах структурного и модульного программирования, структурного анализа и синтеза, например, система Vantage Team Builder (Cayenne).

Объектно-ориентированные методы и CASE-системы получили массовое использование с начала 90-х годов. Они позволяют сократить сроки разработки, а также повысить надежность и эффективность функционирования ИС. Примерами объектно-ориентированных CASE-систем являются Rational Rose (Rational Software) и Object Team (Cayenne).

Комбинированные инструментальные средства поддерживают одновременно структурные и объектно-ориентированные методы, например: Designer/2000 (ORACLE).

По *степени независимости от СУБД* CASE-системы можно разделить на две группы:

- независимые системы;
- встроенные в СУБД.

Независимые CASE-системы поставляются в виде автономных систем, не входящих в состав конкретной СУБД. Обычно они поддерживают несколько форматов баз данных через интерфейс ODBC. К числу независимых CASE-

систем относятся S-Designor (SDP, Powersoft), ERwin (LogicWorks) и Silverrun (Computer Systems Advisers Inc.).

Встроенные CASE-системы обычно поддерживают главным образом формат баз данных СУБД, в состав которой они входят. При этом возможна поддержка и других форматов баз данных. Примером встроенной системы является Designer/2000, входящая в состав СУБД ORACLE.

Рассмотрим наиболее популярные CASE-системы.

7.6. Системы структурного типа

При рассмотрении представителей CASE-систем структурного типа можно выделить две основные группы: независимые и встроенные системы.

Независимые системы

К независимым CASE-системам структурного типа можно отнести популярные продукты S-Designor (фирмы SDP, приобретенной Powersoft), пакет ERwin (LogicWorks) и Silverrun (Computer Systems Advisers Inc.).

S-Designor представляет собой графический инструмент, позволяющий в определенной степени автоматизировать процесс проектирования реляционных БД. Начиная с версии S-Designor 6.0, продукт выпускается под названием PowerDesigner 6.0.

При разработке структуры БД с помощью S-Designor формируется концептуальная модель данных (КМД), которая впоследствии преобразуется в физическую модель данных (ФМД).

Для описания концептуальной модели данных предоставляются удобные средства графического интерфейса в стиле MS Windows. Концептуальная модель данных представляет собой схему базы данных в виде ER-модели.

Сущность изображается прямоугольником, внутри которого расположены атрибуты. Атрибуты, которые однозначно идентифицируют сущность (идентификаторы сущностей), выделяются подчеркиванием. Связи сущностей изображаются линиями, соединяющими соответствующие прямоугольники. Виды связей (1:1, 1:M, M:1, M:M) и подчиненность сущностей отмечаются на окончаниях линий. Если связь имеет место для всех элементов сущности, то линия перечеркивается, в противном случае – вместо перечеркивания изображается кружок. Пример концептуальной модели в виде диаграммы сущностей приведен на рис. 7.7.

При построении концептуальной модели данных можно задать правила контроля ограничений, накладываемых на столбец таблицы (минимальное и максимальное значения, умалчиваемое значение и список допустимых значений).

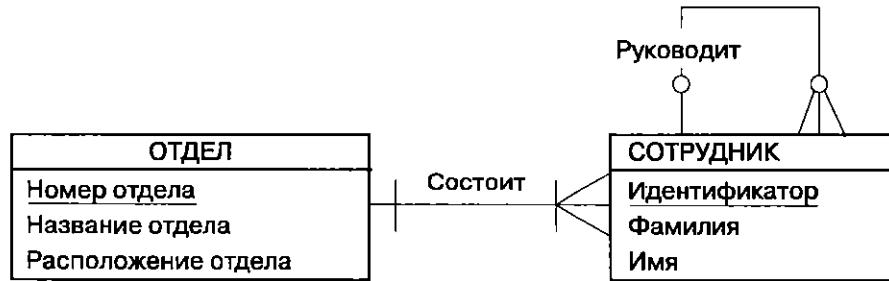


Рис. 7.7. Пример концептуальной модели

Построение физической модели данных проводится на основе концептуальной модели и означает создание таблиц и описаний структур БД для некоторой СУБД или построение готового приложения в специальной среде разработки, например PowerBuilder.

При генерации физической модели данных каждой сущности ставится в соответствие таблица, атрибуты сущностей преобразуются в колонки таблиц, а идентификаторы сущностей становятся ключами.

Если в концептуальной модели данных между сущностями имеется связь вида М:М, то при построении физической модели автоматически создается дополнительная таблица. Ее назначение – нормализация отношения. Колонками таблицы являются идентификаторы участвующих в связи сущностей. Первичный ключ новой таблицы объединяет колонки первичных ключей двух исходных связанных таблиц. Пример перехода от концептуальной модели данных к физической модели данных для связей вида М:М приведен на рис. 7.8. Символьная конструкция вида <pk> обозначает, что эта колонка (поле) таблицы является ключевой.

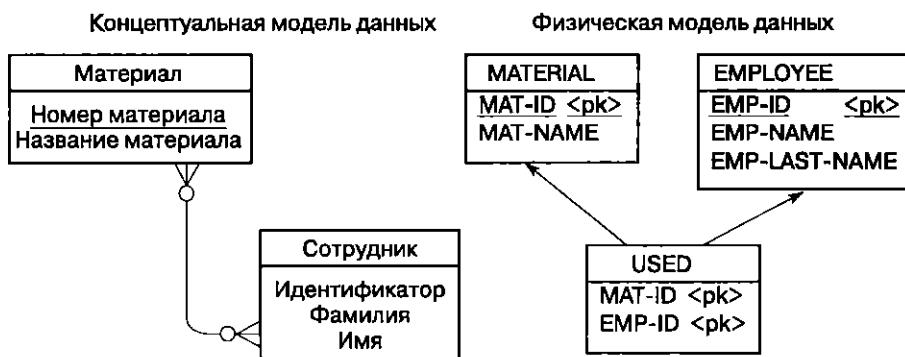


Рис. 7.8. Пример перехода к физической модели

Рассматриваемая система позволяет создавать базы данных путем подключения к работающему серверу СУБД через интерфейс ODBC или готовить текстовые файлы (пакеты) SQL-операторов по созданию структуры БД. Файлы SQL-операторов после этого обрабатываются некоторой СУБД, в результате чего создаются нужные БД.

S-Designor имеет интерфейсы со многими СУБД, включая Oracle, Ingress, Informix, Sybase, SQL Server, Access и Paradox.

Система S-Designor работает в среде Windows и обеспечивает возможность использования других инструментальных средств разработки программ, таких как PowerBuilder, TeamWindows и Progress. Для инструментальной системы PowerBuilder пакет S-Designor позволяет выполнить автоматическую генерацию приложения.

С помощью средств моделирования структур БД системы S-Designor можно осуществлять прямой (к функциональной модели) и обратный (к концептуальной модели) переходы. «Обратное проектирование» информационной модели может понадобиться при решении задач анализа и расширения функций существующих АИС.

Создание таблиц БД сопровождается синтезом средств обеспечения поддержки ссылочной целостности данных в соответствии с типом конкретной СУБД.

Наряду с синтезируемыми программными объектами, система S-Designor поддерживает генерацию отчетов о концептуальной и физической моделях данных. Отчеты можно готовить в виде ASCII-текстов или в формате RTF для текстовых процессоров, например MS Word.

Система S-Designor поддерживает групповую разработку. Модели данных проектируемой информационной системы могут разбиваться на подмодели, с каждой из которых может работать отдельный разработчик. Подмодели данных для удобства хранятся в базах данных. Для хранения моделей может использоваться любая SQL-СУБД. В системе S-Designor имеются средства администрирования групповой работы с парольной защитой.

ERwin представляет собой систему концептуального моделирования баз данных. Система ERwin реализует проектирование схемы БД, генерацию ее описания на языке целевой СУБД (Oracle, Sybase, MS SQL Server и др.) и реинжиниринг баз данных. Для ряда систем быстрой разработки приложений (PowerBuilder, SQL Windows, Delphi, Visual Basic) обеспечивается генерация форм и прототипов приложений.

По функциональным возможностям ERwin близок к S-Designor. В ERwin связь с СУБД организуется напрямую, в отличие от S-Designor, в которой связь с СУБД осуществляется через ODBC-интерфейс с использованием внешних файлов. Отсюда следует, что ERwin менее универсальна и поддерживает меньшее число СУБД.

Silverrun представляет собой открытую систему, используемую совместно с продуктами других различных фирм. Она имеет интерфейсы с СУБД, в число которых входят следующие: DB2, Informix, Ingress, Oracle, Progress, SQLBase, SQLServer. Система Silverrun имеет интерфейсы с системами программирования для языков четвертого поколения (4GL), включая системы PowerBuilder, Progress, SQLWindows, Uniface.

Область применения системы Silverrun – инструментальная поддержка структурных методологий информационных систем бизнес-класса. Эта система ориентирована на начальные стадии проектирования и может быть использована специалистами по анализу и моделированию деятельности организаций, разработчиками информационных систем, а также администраторами БД.

Она позволяет независимо строить модели двух видов: функциональные и информационные. Функциональные модели в виде диаграмм потоков данных DFD ориентированы на пользователей-заказчиков для обоснования требований и постановки задач. Информационные модели в виде диаграмм сущность-связь служат для генерации схем баз данных.

Система Silverrun состоит из трех основных подсистем: модуля построения диаграмм потоков данных и двух модулей построения диаграмм типа сущность-связь: модуля концептуальных моделей ERX (Entity Relationship eXpert) и модуля реляционных моделей RDM (Relational Data Modeler).

Средства построения DFD-диаграмм предоставляют следующие возможности: выбирать вид нотации DFD; изменять внешний вид элементов диаграмм; выбирать набор правил, проверяемых процедурой анализа корректности модели и т. д. Для удобства анализа и реинжиниринга бизнес-процессов предоставляется возможность указывать в моделях объем и удельную стоимость ресурсов, используемых процессами. При этом обеспечивается автоматический подсчет стоимости каждого процесса и общей стоимости определенного ресурса.

В средствах построения концептуальных моделей системы Silverrun, реализуемых модулем ERX, имеется встроенная экспертная система. Последняя помогает реструктурировать не всегда полную и корректную исходную информацию о создаваемой системе к виду, допускающему разработку на ее основе реляционной БД.

Создаваемые с помощью системы Silverrun спецификации схем БД можно переносить в среду окончательной разработки приложения с помощью моста или путем создания файла с SQL-операторами. В первом случае система передает в базу хранения информации о проекте (репозиторий) сведения о форматах ввода, правилах редактирования, формах представления данных и другую информацию. Это означает, что для создания макета приложения остается скомпоновать его, выполнить настройку и корректировку.

В целом система Silverrun по своим возможностям близка к системам S-Designer и ERwin. Система поддерживает коллективную разработку в разнородной среде и может функционировать на платформах Windows, OS/2, Macintosh и Solaris.

Система Designer/2000

CASE-система Designer/2000 фирмы ORACLE является встроенной и используется в СУБД Oracle. Основу CASE-технологии, реализованной в продуктах фирмы ORACLE, составляют:

- методология структурного нисходящего проектирования;
- поддержка всех этапов жизненного цикла прикладной системы;
- ориентация на технологию «клиент-сервер»;
- наличие централизованной базы данных (репозитория) для хранения всей информации в ходе проектирования;
- возможность одновременной работы с репозиторием многих пользователей;
- автоматизация последовательного перехода между этапами разработки;
- автоматизация проектирования и создания приложения (создание документации, проверка спецификаций, автоматическая генерация программ и т. д.).

Система Designer/2000 поддерживает следующие этапы разработки прикладных систем: моделирование и анализ деятельности организации, разработку концептуальных моделей предметной области, проектирование приложения и синтез программ.

Средства поддержки этапа *моделирования и анализа* позволяют строить наядные модели технологических и организационных процессов и структур организации для изучения и совершенствования. При этом широко применяются средства мультимедиа, включая звуковое сопровождение, видео и анимацию.

Модель деятельности организации представляется в виде совокупности диаграмм, описывающих отдельные процессы. Диаграммы строятся из стандартных элементов, основными из которых являются: базовый процесс, шаг процесса, хранилище, поток, организационные единицы и события.

На этапе *концептуального моделирования* предметной области строятся модели, описывающие особенности предметной области, характер решаемых задач, информационные потребности и ресурсы, технологические ограничения и т. д. Используются модели двух видов: информационные (отражают существующие информационные структуры и взаимосвязи между ними) и функциональные (отражают технологию и способы обработки информации).

Основой информационных моделей является специальный вид модели Чена, близкий к бинарной модели типа сущность-связь. В этой модели взаимосвязи могут быть определены между двумя сущностями и взаимосвязи не имеют атрибутов.

Функциональное описание предметной области производится с помощью диаграмм иерархии функций и моделей потоков данных. Первый вид моделей предполагает декомпозицию общей функции на подфункции, каждая из которых, в свою очередь, раскладывается на более мелкие функции и т. д. При необходимости можно описать события, вызывающие выполнение определенной функции. Диаграммы потоков данных позволяют описать движение данных в процессе работы организационных структур.

Концептуальное моделирование в системе Designer/2000 поддерживается совокупностью графических редакторов: ER-диаграмм, иерархии функций и диаграмм потоков данных. Кроме представления моделей, редакторы позволяют вводить дополнительную информацию об элементах диаграмм, выполнять семантические проверки диаграмм на полноту и корректность, получать отчеты и документы по концептуальному моделированию.

На этапе *проектирования* из полученных концептуальных моделейрабатываются технические спецификации на прикладную систему, описывающие структуру и состав БД, а также набор программных модулей. Создаваемые спецификации разделяются на информационные и функциональные, как и исходные концептуальные модели.

Описание структуры и состава БД включает в себя: перечень таблиц БД, состав столбцов (полей) каждой таблицы, состав ключевых полей, состав индексов, ограничения на значения в столбцах, ограничения целостности и т. д.

Функциональное описание будущего приложения предполагает определение: структуры меню пользовательского интерфейса, экранных форм, отчетов, процедурных модулей и прочее.

Первоначальный вариант спецификаций можно получить, воспользовавшись специальными утилитами.

Этап *проектирования* реализуется с помощью трех редакторов: схем программ, диаграмм взаимосвязей модулей и схем модуля. Перечисленные редакторы, кроме построения диаграмм, позволяют вводить дополнительную информацию об отдельных элементах диаграмм.

На этапе *создания программ* используются генераторы программного кода, которые позволяют автоматизировать этот этап, существенно сократить время разработки, повысить качество и надежность получаемого продукта. Имеющиеся в системе генераторы делятся на две группы: генератор серверной части и генераторы клиентских частей.

Генератор *серверной части* по спецификациям БД автоматически строит тексты программ на языке SQL (операторы определения схем БД, триггеров, хранимых процедур и т. д.). Генераторы *клиентских частей* по полученным спецификациям автоматически синтезируют тексты программных модулей (экранные формы, отчеты, процедуры и прочее). Для каждого типа модулей имеется свой генератор.

Работой генераторов можно управлять путем задания более четырехсот параметров, позволяющих изменять внешнее представление приложения, стили оформления текстов, режимы функционирования и т. д. Кроме того, имеются средства разработки нижнего уровня, позволяющие корректировать тексты полученных программ. Имеются также средства реинжиниринга готового приложения, позволяющие по его готовой версии воссоздать спецификации.

7.7. Объектно-ориентированные системы

Появление объектно-ориентированных CASE-систем вызвано рядом преимуществ объектно-ориентированного подхода перед структурным, основанных на трех важнейших свойствах: инкапсуляции, наследовании и полиморфизме.

Инкапсуляция означает объединение в единое целое данных и алгоритмов (функций и методов) их обработки, а также скрытие данных внутри объектов, что позволяет повысить надежность разрабатываемого программного обеспечения. Свойства *наследования* и *полиморфизма* позволяют ускорить процесс разработки программ, а также упростить адаптацию систем на новые условия применения за счет гибкого механизма наращивания возможностей программ в процессе их разработки.

Областью применения объектно-ориентированных инструментальных систем являются сложные проекты, такие как: создание операционных систем, средств разработки приложений и систем реального времени.

В рамках объектно-ориентированного подхода существует множество моделей описания (нотаций) и методов разработки программных систем.

Современные объектно-ориентированные CASE-системы можно разделить на две основные группы: CASE-средства, поддерживающие несколько объектно-ориентированных моделей, и средства, ориентированные только на один вид моделей. В системах первого типа обычно имеется возможность перехода от одной модели к другой. Иногда в этих системах предоставляется возможность создавать собственные нотации.

Объектно-ориентированная система Rational Rose

Rational Rose представляет собой семейство объектно-ориентированных CASE-систем фирмы Rational Software Corporation, служащее для автоматизации анализа и проектирования ПО, генерации кодов на различных языках и подготовки проектной документации. Кроме того, в его составе имеются средства реинжиниринга программ, обеспечивающие повторное использование программных компонентов в новых проектах. В этой системе используется синтез-методология объектно-ориентированного анализа и проектирования Г. Буча, Д. Рамбо и И. Джекобсона, их унифицированный язык моделирования UML (подраздел 7.4).

Конкретный вариант системы определяется языком, на котором выполняется генерация кодов программ (C++, Smalltalk, PowerBuilder, Ada, SqlWindows и ObjectPro). Основным вариантом системы является Rational Rose/C++, позволяющий генерировать программные коды на C++, подготовливать проектную документацию в виде диаграмм и спецификаций.

В процессе работы с помощью Rational Rose выполняется построение диаграмм и спецификаций, определяющих логическую и физическую структуру модели, ее статические и динамические свойства. В их состав входят следующие диаграммы: классов, состояний, сценариев, модулей и процессов.

Основными компонентами системы являются следующие:

- репозиторий, представляющий объектно-ориентированную БД;
- графический интерфейс пользователя;
- средства просмотра проекта, обеспечивающие перемещение по элементам проекта, в том числе по иерархиям классов и подсистем, переключение между видами диаграмм;
- средства контроля проекта, позволяющие находить и устранять ошибки;
- средства сбора статистики;
- генератор документов, позволяющий формировать тексты выходных документов на основе информации из репозитория.

Кроме того, для каждого языка программирования добавляется свой генератор кода и анализатор для C++, обеспечивающий восстановление модели проекта по исходным текстам программ (реинжиниринг). Средства автоматической генерации кодов программ на C++ на основе логической и физической моделей проекта формируют заголовочные файлы и файлы описаний классов и объектов. Полученный таким образом скелет программы можно дополнить путем непосредственного программирования на C++.

Анализатор кодов C++ позволяет создавать модули проектов по информации, содержащейся в определяемых пользователем исходных текстах программ. Анализатор осуществляет контроль правильности исходных текстов и диагностику ошибок. Полученная в результате модель может использоваться в нескольких проектах.

В результате разработки проекта с помощью Rational Rose формируются следующие диаграммы: классов, состояний, сценариев, модулей и процессов. Кроме того, создаются следующие компоненты:

- спецификации классов, объектов, атрибутов и операций;
- заготовки текстов программ;
- модель программной системы.

Модель программной системы представляет собой текстовый файл, содержащий всю информацию о проекте. Заготовки текстов программ формируются в виде заголовочных файлов и заготовок для методов. Система включает в программные файлы комментарии. В окончательные программы исходные тексты заготовок преобразуются программистами.

7.8. Рекомендации по применению CASE-систем

Анализ характеристик и возможностей большинства современных CASE-систем позволяет сделать следующие выводы.

1. CASE-системы позволяют ускорить и облегчить разработку, повысить качество создаваемых программ и информационных систем. Многие из CASE-систем имеют средства управления коллективной работой над проектом.

2. CASE-системы особенно полезными оказываются на начальных этапах разработки. Они являются необязательной частью инструментария разработчика и пока не могут подменить средства проектирования и разработки в составе СУБД. Одной из основных причин этого является разнообразие средств разработки приложений, программно-аппаратных платформ и методологий проектирования.

3. Предоставляемая многими CASE-системами возможность перехода от концептуальной модели БД к физической и обратно полезна для решения задач анализа, совершенствования и переноса приложений из среды одной СУБД в другую.

4. Большинство современных CASE-систем являются структурными, но благодаря некоторым преимуществам объектно-ориентированных систем последние приобретают все большую популярность, особенно при реализации сложных проектов.

5. Современные CASE-системы ориентированы на квалифицированного пользователя, поскольку для их использования требуется знание теории проектирования баз данных. Так, например, для разработки структуры БД с помощью системы S-Designor информацию о проектируемой информационной системе нужно представить в виде ER-модели.

В зависимости от стоящих перед пользователем задач (разработка схемы БД, реинжиниринг, разработка готового приложения и т. д.), условий разработки и других факторов наилучшей может оказаться та или иная CASE-система. Иногда целесообразно использовать несколько CASE-систем.

Применение нескольких CASE-систем часто позволяет объединить достоинства используемых систем и существенно сократить сроки решения задач исследования или разработки. Для примера приведем схему возможного совместного использования CASE-систем ERWin, BPWin и Rational Rose.

Как показано на рис. 7.9, различные CASE-системы могут взаимодействовать друг с другом напрямую (ERWin и BPWin), либо с помощью дополнительных модулей (Model Mart – средство коллективной разработки, ERWin Translation Wizard – модуль импорта в ERWin моделей, созданных в Rational Rose).

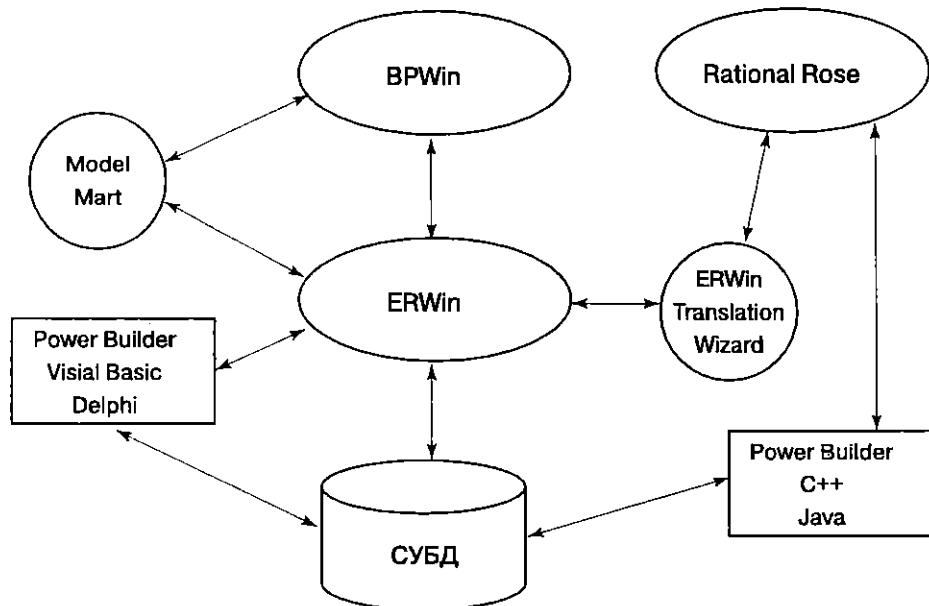


Рис. 7.9. Схема взаимодействия ERWin, BPWin и Rational Rose

Контрольные вопросы и задания

1. Дайте определение CASE-средствам и CASE-технологии.
2. Дайте определение понятия модели жизненного цикла ПО и назовите основные варианты моделей.
3. Перечислите требования к перспективной CASE-системе.
4. Охарактеризуйте спиральную модель жизненного цикла ПО.
5. Перечислите распространенные модели и диаграммы графического представления, используемые при структурном анализе и проектировании.
6. Приведите пример диаграммы потоков данных.
7. Охарактеризуйте методологию функционального моделирования, приведите пример декомпозиции диаграмм.
8. Что представляет собой унифицированный язык моделирования UML?
9. Назовите типы диаграмм унифицированного языка моделирования.
10. Для чего служат диаграммы прецедентов использования и диаграммы классов?
11. Приведите пример диаграммы следования.
12. Назовите признаки классификации CASE-средств.

13. На какие группы делятся CASE-системы по их функциональной ориентации?
14. Приведите пример независимой CASE-системы структурного типа и дайте ей характеристику.
15. Охарактеризуйте CASE-систему Designer/2000.
16. К какому типу CASE-систем относится Rational Rose?
17. Назовите основные компоненты системы Rational Rose.

Литература

1. *Бабкин А., Новиков Ф.* Унифицированный язык моделирования UML. Software Engineering Center St. Petersburg, 1999 (www2.sec.fi/russia/OLD/public/reports).
2. *Вендрев А. М.* CASE-технологии. Современные методы и средства проектирования информационных систем. М.: Финансы и статистика, 1998.
3. *Горин С. В., Тандоев А. Ю.* CASE средство S-Designor 4.2 для разработки структуры базы данных // Системы Управления Базами Данных, № 1, 1996. С. 79–86.
4. *Горчинская О. Ю.* DESIGNER/2000 – новое поколение CASE-продуктов фирмы ORACLE // Системы Управления Базами Данных, № 3, 1995. С. 9–25.
5. *Дило С. М.* Базы данных: проектирование и использование. Учебник. – М.: Финансы и статистика, 2005.
6. *Калянов Г. Н.* CASE. Структурный системный анализ (автоматизация и применение). М.: Лори, 1996.
7. *Коннолли Т., Бегг К.* Базы данных. Проектирование. Реализация и сопровождение. Теория и практика. М.: Вильямс, 2003.
8. *Кумсков М.* Унифицированный язык моделирования (UML) и его поддержка в Rational Rose 98i – CASE-средстве визуального моделирования (www.interface.ru/public).
9. Международные стандарты, поддерживающие жизненный цикл программных средств. М.: МП «Экономика», 1996.
10. *Мюллер Роберт Дж.* Базы данных и UML. Проектирование. – М.: Лори, 2002:
11. *Новоженов Ю. В., Звонкин М. З., Тимонин Н. Н.* Объектно-ориентированные CASE-средства // Системы Управления Базами Данных, № 5–6, 1996. С. 119–125.
12. *Новоженов Ю. В.* Объектно-ориентированные технологии разработки сложных программных систем. М., 1996.
13. *Панащук С. А.* Разработка информационных систем с использованием CASE-системы SILVERRUN // Системы Управления Базами Данных, № 3, 1995. С. 41–47.
14. *Фаулер М., Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. М.: Мир, 1999.

8. Использование баз данных

В разделе рассматриваются важные вопросы применения баз данных: настройка и администрирование, защита информации. Кроме того, описывается характеристика возможностей средств мультимедиа, приобретающих все большее значение.

8.1. Настройка и администрирование

Для успешного функционирования информационной системы, использующей БД, недостаточно выбора СУБД и сервера БД. На начальной стадии запуска информационной системы и в процессе ее эксплуатации необходимо выполнять настройку и различные функции администрирования. Важнейшими задачами администрирования являются защита информации и разграничение доступа пользователей (см. следующий подраздел). К числу других немаловажных задач настройки и администрирования также относятся следующие:

- выбор способа размещения файлов на диске;
- определение требуемого объема дисковой памяти;
- распределение информации на диске;
- резервное копирование.

Рассмотрим коротко решение этих задач.

Выбор способа размещения файлов на диске

Большинство СУБД позволяют администратору системы выбрать один из двух способов размещения файлов БД на дисках: на «чистых» дисках или в *файловой системе* ОС. В первом случае управление данными, хранящимися на отдельных носителях, производится низкоуровневыми средствами самих СУБД. Некоторые СУБД, например Ingres и Interbase, требуют обязательного использования файловой системы (в среде ОС UNIX). Рассмотрим достоинства и недостатки обоих вариантов размещения файлов на диске.

Достоинством хранения информации на «чистых» дисках является то, что *внешняя память* используется более эффективно и, как правило, увеличивается *производительность* обмена с дисками. Экономия внешней памяти от 10 до 20% достигается на основе устранения необходимости организации самой файловой системы. Так, в ОС UNIX служебная информация файловой системы составляет примерно 10% от емкости дисков. Примерно столько же памяти резервируется для последующего возможного расширения файлов. Производительность (обычно порядка 10%) повышается на

основе удаления дополнительного слоя программного обеспечения при обращении к дискам.

Многие СУБД предпочитают работать с дисками через *файловую систему*, которая обеспечивает следующие достоинства. Во-первых, использование файловой системы обладает большей *гибкостью*, так как дает администратору стандартные средства обслуживания файлов: утилиты резервного копирования, архивации, восстановления, а также возможность пользования другими программами работы с файлами (редакторами, антивирусными программами, утилитами контроля качества носителя и т. д.). Во-вторых, в некоторых случаях выполнение операций ввода/вывода через файловую систему обеспечивает *оптимизацию*, которую СУБД не может реализовать. В частности, в системе UNIX происходит кластеризация данных в более крупные физические единицы, чем в большинстве СУБД, что часто приводит к повышению производительности.

Определение требуемого объема дисковой памяти

При определении требуемого объема дисковой памяти следует учитывать, что для обработки данных СУБД использует большой объем служебной информации, размещаемой на дисках. К служебной информации относится следующее: описание схемы базы данных, табличные индексы, временные таблицы, заранее выделенное пространство для хеш-таблиц и индексов, пространство для сортировки, файлы журнала, архивы и многое другое.

Если точная информация об объеме служебной информации для БД отсутствует, то разумно исходить из предположения, что для ее размещения требуется объем дисковой памяти, превосходящий объем собственно данных. Для повышения надежности восстановления данных в базе в случае сбоев и отказов оборудования, файлы журналов транзакций и архивов целесообразно размещать на других физических дисках, отличных от дисков с базой данных.

Распределение информации на дисках

Рациональным и довольно экономичным способом распределения информации на дисках является обеспечение основных задач обработки данных одним или несколькими дисками. Примером такого распределения является использование четырех дисков: одного – для операционной системы и области подкачки (*swap*), другого – для данных, третьего – для журнала транзакций и четвертого – для индексов.

Технически возможно разместить несколько функциональных дисковых областей на одних и тех же (физических или логических) дисках. Так практически всегда делается на СУБД, работающих на ПЭВМ. Однако совместное хранение разнородной информации на одних и тех же дисках приводит к перегрузке подсистемы ввода/вывода и, соответственно, к потере производительности. Перегрузка возникает из-за существенных временных затрат на позиционирование головок на диске.

Поясним, почему увеличивается время на позиционирование головок диска на примере совмещения мест хранения таблицы данных и журнального файла. Пусть приложением выполняется обновление данных базы. Поскольку запись в журнал должна вестись синхронно с обновлением основных данных, головка диска в процессе выполнения операции все время перемещается от области размещения данных к области журнала и обратно.

Резервное копирование

Основным назначением резервного копирования является предотвращение возможной гибели базы данных. Резервное копирование состоит в создании резервной копии базы данных. Напомним, что резервировать нужно не только собственно данные, но и служебную информацию (словарь данных, журнал транзакций и т. д.) СУБД.

При создании резервной копии БД обычно используют магнитные ленты (МЛ) или диски. В последнем случае говорят о так называемом зеркалировании дисков. Резервная копия может быть точной копией исходной БД или сжатой (архивной) копией. Сжатие может осуществляться аппаратно или программно. Аппаратное сжатие предпочтительно с точки зрения временных затрат на эту операцию, но увеличивает стоимость аппаратной части.

Резервное копирование может осуществляться во время работы с БД (режиме *online*) или в другое время. Копия может создаваться по инициативе оператора, либо автоматически в заданное время путем запуска соответствующей утилиты.

При организации резервного копирования перед администратором встают следующие вопросы:

- какие устройства выбрать для резервного копирования;
- когда и с какой частотой выполнять резервное копирование.

1. При определении типа устройства для хранения резервной копии (магнитную ленту или диск) исходят прежде всего из резерва времени, отводимого на процедуру копирования. Для больших БД и медленных устройств время создания копий может быть очень большим. Так, например, для базы данных объемом 20 Гбайт, сбрасываемой на 4 мм магнитную ленту,ирующую со скоростью 500 Кбайт/с, потребуется около 12 часов. Другими важными факторами являются обеспечение согласованного состояния базы данных (за это время постоянно используемая БД будет претерпевать изменения) и время приведения системы в готовность к использованию при восстановлении БД из резервной копии.

Выпускаемые в настоящее время накопители на МЛ позволяют копировать данные со скоростью примерно 1,25 Гбайт в час. Некоторые НМЛ на 8 мм ленте с аппаратной компрессией способны обеспечивать скорость до

3 Гбайтов/час. Некоторые из устройств имеют емкость до 25 Гбайтов, но более типичной является емкость 8–10 Гбайтов.

Отдельные устройства, например Fujitsu Model 2480, могут обеспечивать и большую скорость – около 10 Гбайтов в час, но имеют ограниченный объем носителя (200–400 Мбайт до сжатия). Более современные устройства, например фирмы Metrum, обладают большей пропускной способностью (15 Мбайт/с) и емкостью (14 Гбайтов на одну ленту без компрессии).

Резервное копирование и восстановление с зеркальных дисков происходит намного быстрее. Примером программного обеспечения, выполняющего эти функции с небольшими накладными расходами, является Online:DiskSuite.

2. Резервную копию создают в режиме *online* в случае, если работа с БД происходит круглосуточно, либо когда в работе системы имеются временные окна, продолжительность которых достаточна для создания копии. В остальных случаях создавать копию лучше в конце рабочего дня или недели.

При копировании в режиме *online* возникает проблема обеспечения согласования таблиц БД, поскольку параллельно с получением копии идет работа с данными. Для ее решения все обновления базы данных СУБД «собирает» без обновления таблиц до тех пор, пока не завершится полное копирование, либо пользователям базы данных разрешается доступ только по чтению. Если СУБД не обеспечивает согласованности данных во время резервного копирования, эту функцию должен взять на себя администратор БД.

Важно периодически проверять корректность выполненного резервирования важной информации путем пробного восстановления. Следует не забывать о документировании резервирования.

8.2. Защита информации

Проблему защиты информации в базах данных целесообразно рассматривать совместно с проблемой защиты вычислительной системы (ВС) в целом. Действительно, средой функционирования СУБД – основного инструмента управления данными, является среда вычислительной системы. Кроме того, известные из литературы методы и средства защиты программ и данных в равной мере относятся к программам (СУБД, приложения, хранимые процедуры и т. д.) и данным (базы данных, словари данных) из баз данных.

Знание принципов построения систем защиты и возможностей, предоставляемых различными компонентами вычислительной системы (операционной системой, программами обслуживания, СУБД, специализированными пакетами защиты и отдельными устройствами) позволяет оценить уязвимость ИС и грамотно организовать в ней защиту конфиденциальной информации.

Основные определения

Информация требует защиты на автономном компьютере и на машине, соединенной с другими отдельными компьютерами или подключенной к локальной или глобальной сети. Поскольку сетевой режим является более общим случаем использования компьютеров, при обсуждении вопросов защиты информации предположим наличие сетевых связей между компьютерами.

Под **информацией** (информационным обеспечением) в ВС понимается любой вид накапливаемых, хранимых и обрабатываемых данных. Частным случаем информации является совокупность программ (системных и прикладных), обеспечивающих различную обработку данных.

Движение информации в ЭВМ неразрывно связано с функционированием программ ее обработки и обслуживания, поэтому при рассмотрении защиты информации в ВС говорят об информационно-программном обеспечении (ИПО).

Целью защиты ИПО является обеспечение безопасности хранимой и обрабатываемой информации, а также используемых программных средств.

Для обеспечения качественной защиты ВС недостаточно организовать охрану помещений и установить программные средства защиты. Здесь требуется комплексный (системный) подход, предполагающий организацию защиты вычислительной системы в соответствии со структурой ВС, задачами, решаемыми в ней, а также целями и возможностями защиты.

Систему защиты нужно закладывать с начала разработки ВС и ее ИПО и выполнять на всех этапах жизненного цикла. На практике, в силу ряда причин, создание системы защиты часто начинается когда ВС и ее ИПО уже разработаны. Создаваемая система защиты должна быть многоуровневой, адаптируемой к новым условиям функционирования, включать в себя рационально организованную совокупность имеющихся средств, методов и мероприятий. Защита должна быть от злоумышленников и от некомпетентных действий пользователей и обслуживающего персонала.

Для эффективного построения системы защиты необходимо:

- выделить уязвимые элементы вычислительной системы;
- выявить угрозы для выделенных элементов;
- сформировать требования к системе защиты;
- выбрать методы и средства удовлетворения предъявленным требованиям.

Безопасность ВС нарушается вследствие реализации одной или нескольких потенциальных угроз. Под **угрозой** ИПО понимается возможность преднамеренного или случайного действия, которое может привести к нарушению безопасности хранимой и обрабатываемой в ВС информации, в том числе и программ.

Основными видами угроз в ВС являются следующие:

1. Несанкционированного использования ресурсов ВС:
 - использование данных (копирование, модификация, удаление, печать и т. д.);
 - копирование и модификация программ;
 - исследование программ для последующего вторжения в систему.
2. Некорректного использования ресурсов ВС:
 - случайный доступ прикладных программ к чужим разделам основной памяти;
 - случайный доступ к системным областям дисковой памяти;
 - некорректное изменение баз данных (ввод неверных данных, нарушение ссылочной целостности данных);
 - ошибочные действия пользователей и персонала.
3. Проявления ошибок в программных и аппаратных средствах.
4. Перехвата данных в линиях связи и системах передачи.
5. Несанкционированной регистрации электромагнитных излучений.
6. Хищения устройств ВС, носителей информации и документов.
7. Несанкционированного изменения состава компонентов ВС, средств передачи информации или их вывода из строя и т. д.

*Возможными **последствиями нарушения защиты** являются следующие:*

- получение секретных сведений;
- снижение производительности или остановка системы;
- невозможность загрузки операционной системы с жесткого диска;
- материальный ущерб;
- катастрофические последствия.

Целью защиты является обеспечение безопасности информации в ВС, которая может быть нарушена (случайно или преднамеренно), поэтому сущность защиты сводится к предотвращению угроз нарушения безопасности.

Исходя из возможных угроз безопасности можно выделить три основные **задачи защиты**:

- защита информации от хищения;
- защита информации от потери;
- защита ВС от сбоев и отказов.

Защита информации от хищения подразумевает предотвращение физического хищения устройств и носителей хранения информации, несанкционированного получения информации (копирования, подсмотра, перехвата и т. д.) и несанкционированного распространения программ.

Защита информации от потери подразумевает поддержание целостности и корректности информации, что означает обеспечение физической, логической и семантической целостности информации. Информация в системе может быть потеряна как из-за несанкционированного доступа в систему поль-

вателей, программ (в том числе и компьютерных вирусов), некорректных действий пользователей и их программ, обслуживающего персонала, так и в случаях сбоев и отказов в ВС.

Защита от сбоев и отказов аппаратно-программного обеспечения ВС является одним из необходимых условий нормального функционирования системы. Если вычислительная система является ненадежной, информация в ней часто искажается и иногда утрачивается. Основная нагрузка на обеспечение хорошей защиты от сбоев и отказов в системе ложится на системные аппаратно-программные компоненты: процессор, основную память, внешние запоминающие устройства, устройства ввода-вывода и другие устройства, а также программы операционной системы. При недостаточно надежных системных средствах защиту от сбоев следует предусматривать в прикладных программах.

Под *надежностью* ПО понимается способность точно и своевременно выполнять возложенные на него функции. Степень надежности ПО определяется качеством и уровнем автоматизации процесса разработки, а также организацией его сопровождения. Так как достичь 100%-й надежности программ на практике почти не удается, необходимо предусматривать средства быстрого восстановления работоспособности программ и данных после восстановления аппаратуры и ПО от сбоев и отказов.

Для организации комплексной защиты информации в ВС в общем случае может быть предусмотрено *4 защитных уровня*.

1. Внешний уровень, охватывающий всю территорию расположения ВС.
2. Уровень отдельных сооружений или помещений расположения устройств ВС и линий связи с ними.
3. Уровень компонентов ВС и внешних носителей информации.
4. Уровень технологических процессов хранения, обработки и передачи информации.

Первые три уровня обеспечивают в основном физическое препятствие доступу путем ограждения, системы сигнализации, организации пропускного режима, экранирования проводов и т. д. Последний уровень предусматривает логическую защиту информации в том случае, когда физический доступ к ней имеется.

Методы и средства защиты

Существующие методы защиты можно разделить на четыре основных класса:

- физические;
- аппаратные;
- программные;
- организационные.

Физическая защита используется в основном на верхних уровнях защиты и состоит в физическом преграждении доступа посторонних лиц в помеще-

ния ВС на пути к данным и процессу их обработки. Для физической защиты применяются следующие средства:

- сверхвысокочастотные, ультразвуковые и инфракрасные системы обнаружения движущихся объектов, определения их размеров, скорости и направления перемещения;
- лазерные и оптические системы, реагирующие на пересечение нарушаителями световых лучей;
- телевизионные системы наблюдения за охраняемыми объектами;
- кабельные системы, в которых небольшие объекты окружают кабелем, чувствительным к приближению нарушителя;
- системы защиты окон и дверей от несанкционированного проникновения, а также наблюдения и подслушивания;
- механические и электронные замки на двери и ворота;
- системы нейтрализации излучений.

Аппаратная защита реализуется аппаратурой в составе ЭВМ или с помощью специализированных устройств. Основными аппаратными средствами защиты являются средства защиты процессоров и основной памяти, устройств ввода-вывода, систем передачи данных по каналам связи, систем электропитания, устройств внешней памяти (зеркальные диски) и т. д.

Аппаратные средства защиты процессоров производят контроль допустимости выдаваемых из программ команд. Средства защиты памяти обеспечивают режим совместного использования и разграничения оперативной памяти при выполнении программ. К аппаратным средствам защиты устройств ввода-вывода относятся различные схемы блокировки от несанкционированного использования. Средства защиты передачи данных по каналам связи представляют собой схемы засекречивания (шифрования) информации.

Программная защита реализуется с помощью различных программ: операционных систем, программ обслуживания, антивирусных пакетов, инструментальных систем (СУБД, электронных таблиц, текстовых процессоров, систем программирования и т. д.), специализированных программ защиты и готовых прикладных программ.

Организационная защита реализуется совокупностью направленных на обеспечение защиты информации организационно-технических мероприятий, разработкой и принятием законодательных актов по вопросам защиты информации, утверждением морально-этических норм использования информации в обществе и т. д.

Остановимся более подробно на наиболее гибких и мощных методах защиты – программно-аппаратных методах, реализуемых в ВС.

Программно-аппаратные методы защиты

С помощью программно-аппаратных средств можно в определенной мере решать как основные задачи защиты ИПО в ВС (от хищения, от потери, от сбоев и отказов оборудования), так и защиту от ошибок в программах.

Решение этих задач в системах защиты обеспечивается следующими способами:

- 1) защитой от несанкционированного доступа (НСД) к ресурсам со стороны пользователей и программ;
- 2) защитой от несанкционированного использования (НСИ) ресурсов при наличии доступа;
- 3) защитой от некорректного использования ресурсов;
- 4) внесением структурной, функциональной и информационной избыточности;
- 5) высоким качеством разработки программно-аппаратных средств.

Рассмотрим перечисленные способы более подробно и укажем методы их реализации.

1. Для защиты от НСД прежде всего необходима эффективная *система регистрации* попыток доступа в систему со стороны пользователей и программ, а также мгновенная *сигнализация* о них отвечающим за безопасность ВС лицам. Именно отсутствие надежной системы регистрации и сигнализации при НСД, а также наличие обходных путей или «дыр» в ВС, является причиной незаконного проникновения в систему. Чтобы регистрировать события подключения к системе, в ВС обычно ведется специальный журнал или база данных.

Защита от НСД со стороны пользователей в современных системах в основном реализуется двумя основными способами: парольной защитой, а также путем идентификации и аутентификации.

Простейшая *парольная защита* является достаточно слабым средством, особенно если пароль не шифруется. Основной ее недостаток состоит в том, что все пользователи, использующие одинаковый пароль, с точки зрения ВС неразличимы. Неудобство парольной защиты для пользователя состоит в том, что надо запоминать пароль. Если он простой и короткий, значит, его легко подобрать, если сложный – его нужно куда-нибудь записать. При небрежном отношении к записям пароль может стать достоянием других.

Для получения доступа к ВС достаточно знать некоторый пароль. После ввода этого пароля обычно разрешается все. Иногда в системе имеется несколько паролей, за каждым из которых закреплены соответствующие права доступа.

Более серьезный контроль доступа в систему получается, если каждого подключающегося пользователя сначала идентифицировать, затем убедиться, что это именно он, а не другой (аутентифицировать), и при запросе ресурсов контролировать полномочия (роверять право запрашивать ресурсы системы).

Идентификация пользователей может выполняться, например, с помощью паролей. Для *аутентификации*, или проверки подлинности пользователя, часто используют следующие способы:

- запрос секретного пароля;
- запрос какой-либо информации сугубо индивидуального;

- проверка наличия физического объекта, представляющего собой электронный аналог обычного ключа (электронный ключ);
- применение микропроцессорных карточек;
- активные средства опознавания;
- биометрические средства.

Запрашиваемая для аутентификации *дополнительная информация* может представлять собой любые данные, связанные с некоторыми сведениями, явлениями или событиями из личной жизни пользователя или его родственников. Например, номер счета в банке, номер технического паспорта автомобиля, девичья фамилия матери или жены и т. д.

Примером *электронного ключа* является пластиковая карточка с магнитной полоской. На запоминающем слое хранится код, выполняющий роль не-видимого пароля. Более сложный вариант электронного ключа – специальный прибор, называемый *жетоном* и позволяющий генерировать псевдослучайные пароли.

Существуют различные варианты реализации жетонов. Одним из первых довольно удачных решений является жетон SecurID американской фирмы Security Dynamics, появившийся в 1987 году. В нем генерируемая случайным образом буквенно-цифровая последовательность (пароль) меняется примерно раз в минуту синхронно с паролем в центральной части системы защиты. Это значит, что каждый новый пароль имеет ограниченное время действия. Сами пароли постоянно изменяются, усложняя подбор со стороны злоумышленников. Каждый пароль при этом пригоден для однократного входа в систему. Жетоны SecurID популярны и в настоящее время как средства аутентификации пользователей. Другим интересным решением является устройство фирмы SmartDisk Security, использующее 3.5-дюймовый дисковод и дискеты в качестве жетонов.

Одним из *недостатков* жетонов является проблема обеспечения доступа к системе в случае, если жетон забыт дома. Выход в такой ситуации находят в создании некоторого количества временных жетонов.

Недавно появившиеся на рынке микропроцессорные карточки, разработанные Национальным институтом стандартов и технологий США, позволяют формировать цифровые подписи. Алгоритм шифрования обеспечивает невозможность подделки электронных подписей.

Более перспективными средствами аутентификации являются так называемые *активные средства опознавания*. Примером такого средства является система, состоящая из миниатюрного слабосигнального радиопередатчика и соответствующего радиоприемника. При подключении к системе пользователь должен приблизить на небольшое расстояние (порядка нескольких дециметров) к приемнику передатчик и включить его. Если принятый сигнал опознается, пользователь получает доступ к системе. *Достоинство* такой системы – отсутствие физического контакта.

Из множества существующих средств аутентификации наиболее надежными (но и дорогими) считаются биометрические средства. В них опознание личности осуществляется по отпечаткам пальцев, форме ладони, сетчатке глаза, подписи, голосу и другим физиологическим параметрам человека. Некоторые системы идентифицируют человека по манере работы на клавиатуре. Основным достоинством систем такого класса является высокая надежность аутентификации. Недостатки систем включают в себя высокую стоимость оборудования, временную задержку распознавания некоторых средств (десятка секунд) и неудобство для пользователя.

Уверенность в том, что подключающийся к системе пользователь или программа, не являются злоумышленными, не дает гарантии безопасности последующего поведения во время работы, поэтому во многих системах защиты предусматривается *разграничение доступа к ресурсам* в течение сеанса.

По завершении сеанса работы информация о параметрах подключения, в том числе пароли, в вычислительной системе должна удаляться, чтобы ею не могли воспользоваться несанкционированные программы и пользователи. Если же «прощание с системой после реального прекращения работы затянулось» (это может быть забывчивость пользователя выполнить процедуру отключения или некорректное завершение работы программы), система защиты должна предусматривать механизмы принудительного завершения работы и закрытия каналов доступа от посторонних пользователей и программ. Отключение объектов от ВС можно выполнять, например, после анализа их активности в течение некоторого времени, отсутствия ответов на предупреждения об отключении пользователей, либо по истечении продолжительности сеанса работы.

Одной из разновидностей несанкционированных программ являются компьютерные вирусы. Количество известных компьютерных вирусов постоянно возрастает. Появилась даже новая инженерная дисциплина – компьютерная вирусология. Последствия воздействия компьютерных вирусов могут быть разнообразными: от внешне необычных эффектов на мониторе компьютера и простого замедления работы ЭВМ до краха вычислительной системы или сети. Отсюда возникает необходимость *защиты от компьютерных вирусов* на всех стадиях их развития и в особенности на стадиях их проникновения в систему и размножения. Для этого в систему защиты включают средства диагностирования состояния программно-аппаратных средств, локализации и удаления вирусов, устранения последствий их воздействия.

2. Обеспечение защиты от НСИ ресурсов, как и от НСД, требует применения *средств регистрации* запросов запицаемых ресурсов ВС и *сигнализации* в случаях попыток незаконного их использования. Заметим, что речь ведется о важнейших с точки зрения защиты ресурсах. Если постоянно регистрировать все события обо всех запросах на ресурсы в ВС, на остальную работу не хватит процессорного времени.

Для защиты информационно-программных ресурсов ВС от несанкционированного использования применяются следующие варианты защиты: *от копирования, исследования (программ), просмотра (данных), модификации и удаления*. Приведем примеры их применения.

Для защиты программы от несанкционированного *копирования* можно в исполняемом коде выполнить привязку к оборудованию. Тогда копия программы не будет работать на другом компьютере.

Под защитой от *исследования* программ понимаются такие средства, которые не позволяют или затрудняют изучение системы защиты программы. Например, после нескольких неудачных попыток подключения к программе, имеющей парольную защиту, целесообразно блокировать дальнейшие попытки подключения к ней либо предусмотреть средства самоликвидации.

Защиту файлов с исполняемыми программами или данными от *модификации* можно сделать путем сверки некоторой характеристики файла (контрольной суммы) с эталоном. Тогда, если кто-нибудь изменит содержимое файла, изменится его контрольная сумма, что сразу же обнаружится. Средства проверки контрольной суммы можно вставить в программу (для программных файлов) либо поместить в программную систему контроля модификации файлов (программ и данных).

Защитить от *удаления* программы или данные можно путем предотвращения несанкционированных операций удаления файлов в вычислительной системе. К сожалению, широко распространенные операционные системы MS DOS и MS Windows стандартных эффективных средств такого рода не имеют. С этой целью можно разработать или подобрать из имеющихся резидентную программу контроля функции удаления файла с диска.

Достаточно мощным средством защиты данных от *просмотра* является их *шифрование*. Расшифровка информации требует знания ключа шифрования. Подбор последнего даже при современном уровне компьютерной техники представляет трудоемкую задачу.

Шифрование незаменимо для защиты информации от раскрытия ее содержания при хранении информации в файлах или базах данных, а также при передаче по линиям связи: проводным, кабельным и радиоканалам.

Шифрование данных осуществляется в темпе поступления информации (On-Line) и в автономном режиме (Off-Line). Первый способ применяется в основном в системах приема-передачи информации, а второй – для засекречивания хранимой информации.

В современных системах защиты в основном применяется два алгоритма: DES и RSA. Коротко их рассмотрим.

Стандарт шифрования данных – Data Encryption Standard (DES) разработан фирмой IBM в начале 70-х годов, рекомендован Ассоциацией Американских Банкиров и является правительственным стандартом цифрового шифрования.

В алгоритме DES используется ключ длиной 56 бит и 8 бит проверки на четность. Он обеспечивает высокую степень защиты при небольших расходах на шифрование, требуя для подбора ключевой комбинации перебора 72 квадриллионов вариантов.

Алгоритм DES является *симметричным* в том смысле, что для шифрования и дешифрования некоторой информации он использует один и тот же ключ. Если в процессе функционирования вычислительной сети между корреспондентами необходимо передать полномочия по шифрованию, то передаваемые для этого ключи шифрования необходимо засекречивать (шифровать). Длина ключа и контрольных битов для алгоритма фиксированы.

Другой алгоритм – RSA (сокращение по фамилиям авторов) предложен Ривестом, Шамиром и Альдеманом в 1976 году. Алгоритм является более совершенным и принят в качестве стандарта Национальным Бюро Стандартов.

В алгоритме RSA используются различные ключи для шифрования и дешифрования, т. е. он является *асимметричным*. Поскольку ключ для шифрования не годится для дешифрации, его можно смело передавать по сети, а поэтому ключ шифрования часто называют *открытым* ключом.

Достоинством алгоритма RSA является также то, что он работает при разной длине ключа. Чем длиннее ключ, тем большее время требуется на выполнение операции преобразования информации и тем выше уровень безопасности.

Алгоритмы шифрования реализуются программно или аппаратно. Одним из примеров аппаратной реализации является сравнительно недорогая микросхема шифрования Clipper (не путать с системой программирования Clipper для баз данных). Система, разработанная на базе этой микросхемы, предназначена для защиты речевой информации. Внедрение ее поддержано Агентством национальной безопасности США.

По завершению работы программы необходимо позаботиться об уничтожении данных из оперативной и внешней памяти. При возникновении серьезной угрозы использования конфиденциальных данных желательно в системе защиты иметь возможность аварийного их удаления.

3. Защита от *некорректного использования ресурсов* традиционно выполняется программами ОС. Функции защиты от некорректного использования ресурсов ВС предусматривают, по крайней мере, следующие действия: *изолирование друг от друга участков оперативной памяти, выделенных различным программам, защиту системных областей внешней памяти и контроль допустимости команд ЦП*.

В программном обеспечении на более высоком, чем ОС, уровне необходимо обеспечить корректность использования прикладных ресурсов: документов, изображений, баз данных, сообщений и т. п. На практике возможны ситуации, когда корректные с точки зрения операционной системы файлы содержат не совсем верную или противоречивую информацию из предмет-

ной области. Другими словами, прикладное программное обеспечение тоже должно обеспечивать *целостность и непротиворечивость* данных.

4. Одним из важнейших методов устранения или сведения к минимуму последствий сбоев и отказов в работе ВС является внесение структурной, функциональной и информационной избыточности (резервирования).

Структурная избыточность означает резервирование аппаратных компонентов ВС на различных уровнях: ЭВМ (дублирование серверов обработки); отдельных устройств (дублирование процессоров или накопителей на магнитных дисках – зеркальные диски) и схем устройств (мажоритарные схемы выполнения операций). При резервировании следует обеспечить прежде всего стабильное и бесперебойное питание, к примеру, с помощью источников бесперебойного питания.

Функциональное резервирование означает организацию вычислительного процесса, при которой функции управления, хранения и обработки информации реализуются несколькими элементами системы. При отказе функционального элемента его заменяет другой элемент. Примером функциональной избыточности может служить запуск нескольких одинаковых программ в многозадачной операционной системе.

Информационное резервирование используется для предотвращения полной потери информации и реализуется путем одноразового или периодического копирования и архивирования наиболее ценной информации. К ней прежде всего можно отнести прикладные программы пользователя, а также данные различных видов: документы, БД, файлы и т. д., а также основные программы ОС, типовое ПО (СУБД, текстовые, табличные и графические процессоры и т. п.).

Резервирование информации можно выполнять путем копирования ценной информации на вспомогательные носители информации: жесткие диски, диске́ты, накопители на оптических дисках, магнитные ленты. Более эффективным по расходованию внешней памяти является создание сжатых архивов исходной информации. Получение исходной информации из сжатой выполняется с помощью соответствующего разархиватора или путем запуска на выполнение саморазархивирующегося файла. Иногда при сжатии информации используют парольную защиту, позволяющую восстановить исходную информацию при задании пароля. Практически это удобно, так как решаются две задачи: создается сжатая копия и она защищается от несанкционированного просмотра.

Примером архиватора, позволяющего создавать защищенные паролем архивные файлы, являются программы pkzip и pkunzip фирмы PKWARE Inc. Основной функцией первой из них является создание архива (при этом необходимо указать пароль), а вторая – выполняет полное или частичное восстановление в первоначальный вид.

Своевременное выявление сбоев и отказов оборудования, а также физических и логических дефектов на носителях информации невозможно без

организации тестирования аппаратно-программных средств. *Тестирование* может выполняться в специально отведенное время и в процессе работы (например, в интервалы простоя оборудования).

При выявлении в системе ошибок, требуется проведение восстановительных операций. *Восстановление* искаженных или потерянных данных и программ обычно выполняется после тестирования. В ответственных случаях применяют *самотестирование* и *самовосстановление* программ, при котором перед началом вычислений программа проверяет наличие и корректность исходных данных и при обнаружении ошибок производит восстановление данных.

5. Многие причины потери информации в процессе обычного функционирования системы, а также в результате происходящих в системе сбоев и отказов, кроются в наличии ошибок или неточностей, заложенных на этапах проектирования ВС.

Для устранения или сведения к минимуму ошибок, которые существенно снижают общую защищенность ВС, следует использовать современные методы защиты на всех этапах жизненного цикла аппаратно-программного обеспечения ВС: системного анализа, проектирования, эксплуатации и сопровождения.

Например, при проектировании программного обеспечения широко применяются методы объектно-ориентированного и визуального программирования.

Примеры программных систем защиты

Из существующих программных систем защиты достаточно популярными являются системы «Керберос» (Kerberos) и «Кобра».

Система «Керберос» является системой аутентификации пользователей, разработана в 80-х годах в Массачусетском технологическом институте. Основное ее назначение – контроль доступа пользователей в вычислительной сети. Система эффективно функционирует в распределенных системах с небольшим числом централизованно управляемых рабочих станций. Предполагается, что имеется хорошо защищенная система управления обменом ключами шифрования, рабочие станции не защищены, а серверы имеют слабую защиту. В «Керберосе» применяется многократное шифрование при передаче служебной информации в сети. Так, в некоторых зашифрованных сообщениях, отдельные слова сообщения тоже зашифрованы. Пароли никогда не передаются по сети незашифрованными. При обмене служебной информацией применяются зашифрованное данные, действующие ограниченное время, – аутентикаторы (authenticator), содержащие имя пользователя, его сетевой адрес и отметку времени.

В системе «Керберос» используется алгоритм шифрования RSA. Система «Керберос» может работать на различных платформах, в том числе MS-DOS,

Macintosh, SunOS, HP-UX, NextStep, и AIX-системы RS/6000 фирмы IBM. Совместно с ней могут применяться также жетоны SecurID фирмы Security Dynamics.

Система «Керберос» обладает рядом недостатков. Во-первых, она не позволяет выполнять проверку полномочий прикладных задач и отдельных транзакций. Во-вторых, необходим предварительный обмен ключами шифрования между всеми участниками обмена. В-третьих, в процессе работы элементы системы обмениваются служебной информацией, что требует высокой пропускной способности каналов обмена данными.

Система «Керберос» имеет структуру приложений типа клиент-сервер. Она состоит из двух основных частей: клиентской части (клиента) и серверной части (сервера). «Керберос-сервер» состоит из трех серверов: идентификационного сервера, сервера выдачи разрешения и сервера выполнения административных функций. Вся область защиты он несанкционированного доступа может состоять из нескольких зон, в каждой из которых должен быть свой сервер.

Упрощенно работу системы защиты «Керберос» можно представить следующим образом. Пользователь вводит свой идентификационный код (имя), который шифруется клиентом и направляется к идентификационному серверу как запрос на выдачу «разрешения на получение разрешения». Другими словами, формируется запрос на регистрацию к системе защиты.

Идентификационный сервер отыскивает в своей базе данных разрешенных пользователей соответствующий пароль, с его помощью шифрует ответное сообщение, которое отсылает клиенту. Получив «разрешение на разрешение», клиент расшифровывает его, определяет из него значение пароля пользователя и запрашивает пароль у пользователя. Если введенный и полученный пароли совпадают, клиент формирует шифрованный запрос серверу выдачи разрешения на получение доступа к требуемым ресурсам сети.

После ряда манипуляций (расшифровывания и ряда проверок) и полной уверенности, что подключающийся пользователь тот, за кого себя выдает, сервер выдачи разрешения отсылает пользователю зашифрованное разрешение на доступ к ресурсам сети.

Получив и расшифровав разрешение, клиент связывается с помощью зашифрованного сообщения с целевым сервером, ресурсы которого требуются пользователю, и только после этого пользователь получает доступ к ресурсу.

Для обеспечения еще более высокого уровня защиты клиент может потребовать идентификации целевого сервера, а не безусловной связи с ним. В этом случае устраняется возможность перехвата информации, дающей право на доступ к ресурсам сети.

Система «Кобра» является одной из распространенных и эффективных специализированных систем защиты для MS-DOS и Windows. Она основана

на технологии прозрачной защиты, которой пользователь в своей работе не замечает, а поэтому не испытывает неудобств от функционирования защитных средств.

Прозрачная защита в системе «Кобра» строится с помощью метода динамического шифрования. Конфиденциальная информация, записываемая на внешние устройства, автоматически зашифровывается по ключу, зависящему от пароля пользователя. При считывании зашифрованной информации происходит ее автоматическая дешифрация.

Для шифрования информации в системе «Кобра» применяется технология криптозащиты, обеспечивающая повышение скорости шифрования и криптостойкости зашифрованной информации. Скорость шифрования увеличивается за счет двухэтапной схемы шифрования, а криптостойкость – за счет внесения неопределенности в алгоритм шифрования.

Средства защиты БД

Средства защиты БД в различных СУБД несколько отличаются друг от друга. На основе анализа современных СУБД фирм Borland и Microsoft можно утверждать, что средства защиты БД условно делятся на две группы: основные и дополнительные.

К **основным средствам защиты** информации можно отнести следующие средства:

- парольной защиты;
- шифрования данных и программ;
- установления прав доступа к объектам БД;
- защиты полей и записей таблиц БД.

Парольная защита представляет простой и эффективный способ защиты БД от несанкционированного доступа. Пароли устанавливаются конечными пользователями или администраторами БД. Учет и хранение паролей производится самой СУБД. Обычно пароли хранятся в определенных системных файлах СУБД в зашифрованном виде. Поэтому просто найти и определить пароль невозможно. После ввода пароля пользователю СУБД предоставляются все возможности по работе с защищенной БД. Саму СУБД защищать паролем большого смысла нет.

Шифрование данных (всей базы или отдельных таблиц) применяют для того, чтобы другие программы, «з나ющие формат БД этой СУБД», не могли прочитать данные. Такое шифрование (применяемое в Microsoft Access), по-видимому, дает немного, поскольку расшифровать БД может любой с помощью «родной» СУБД. Если шифрация и дешифрация требуют задания пароля, то дешифрация становится возможной при верном вводе пароля. Надеемся, разработчики Access учтут это в последующих версиях.

Шифрование исходных текстов программ позволяет скрыть от несанкционированного пользователя описание соответствующих алгоритмов.

В целях контроля использования основных ресурсов СУБД во многих системах имеются средства установления *прав доступа* к объектам БД. Права доступа определяют возможные действия над объектами. Владелец объекта (пользователь, создавший объект), а также администратор БД имеют все права. Остальные пользователи к разным объектам могут иметь различные уровни доступа.

По отношению к таблицам в общем случае могут предусматриваться следующие права доступа:

- просмотр (чтение) данных;
- изменение (редактирование) данных;
- добавление новых записей;
- добавление и удаление данных;
- все операции, в том числе изменение структуры таблицы.

К данным, имеющимся в таблице, могут применяться меры защиты по отношению к отдельным полям и отдельным записям. В известных нам реляционных СУБД, отдельные записи специально не защищаются, хотя можно привести примеры из практики, когда это требуется. Контроль прав доступа, по-видимому, должен быть в объектно-ориентированных СУБД, в которых есть идентификация отдельных записей (одно из отличий объектно-ориентированной модели от реляционной).

Применительно к защите данных в полях таблиц можно выделить следующие уровни прав доступа:

- полный запрет доступа;
- только чтение;
- разрешение всех операций (просмотр, ввод новых значений, удаление и изменение).

По отношению к формам могут предусматриваться две основные операции: вызов для работы и разработка (вызов Конструктора). Запрет вызова Конструктора целесообразно делать для экранных форм готовых приложений, чтобы конечный пользователь случайно не испортил приложение. В самих экранных формах отдельные элементы могут быть тоже защищены. Например, некоторые поля исходной таблицы вообще могут отсутствовать или скрыты от пользователя, а некоторые поля – доступны для просмотра.

Отчеты во многом похожи на экранные формы, за исключением следующего. Во-первых, они не позволяют изменять данные в таблицах, а во-вторых, основное их назначение – вывод информации на печать. На отчеты, также как и на экранные формы, может накладываться запрет на вызов средств их разработки.

Для исключения просмотра и модификации (случайной и преднамеренной) текстов программ, используемых в приложениях СУБД, помимо шифрации, может применяться их парольная защита.

К *дополнительным средствам защиты* БД можно отнести такие, которые нельзя прямо отнести к средствам защиты, но которые непосредственно влияют на безопасность данных. Их составляют следующие средства:

- встроенные средства контроля значений данных в соответствии с типами;
- повышения достоверности вводимых данных;
- обеспечения целостности связей таблиц;
- организации совместного использования объектов БД в сети.

Редактируя БД, пользователь может случайно ввести такие значения, которые не соответствуют типу поля, в которое это значение вводится. Например, в числовое поле пытаться занести текстовую информацию. В этом случае СУБД с помощью *средств контроля значений* блокирует ввод и сообщает пользователю об ошибке звуковым сигналом, изменением цвета вводимых символов или другим способом.

Средства повышения достоверности вводимых значений в СУБД служат для более глубокого контроля, связанного с семантикой обрабатываемых данных. Они обычно обеспечивают возможность при создании таблицы указывать следующие ограничения на значения: минимальное и максимальное значения; значение, принимаемое по умолчанию (если нет ввода), требование обязательного ввода; задание маски (шаблона) ввода; указание дополнительной сверочной таблицы, по которой ведется контроль вводимых значений и т. д.

Более совершенной формой организации контроля достоверности информации в БД является разработка хранимых процедур. Механизм хранимых процедур применяется в БД, размещенных на сервере. Сами хранимые процедуры представляют собой программы, алгоритмы которых предусматривают выполнение некоторых функций (в том числе контрольных) над данными. Процедуры хранятся вместе с данными и при необходимости вызываются из приложений либо при наступлении некоторых событий в БД.

Решение прикладной задачи, как правило, требует информации из нескольких таблиц. Сами таблицы для удобства обработки и исключения дублирования информации некоторым образом связываются (подраздел 3.3). Функции поддержания логической целостности связанных таблиц берет на себя СУБД (подраздел 3.4). К сожалению, далеко не все СУБД в полной мере реализуют эти функции, в этом случае ответственность за корректность связей возлагается на приложение.

Приведем пример возможных действий СУБД по контролю целостности связей таблиц. Пусть между двумя таблицами существует связь вида 1:М и, следовательно, одной записи основной таблицы может соответствовать несколько записей вспомогательной таблицы.

При вставке записей во вспомогательную таблицу система контролирует наличие соответствующих значений в поле связи основной таблицы. Если вводимое значение отсутствует в основной таблице, СУБД временно блоки-

рут работу с новой записью и предлагает изменить значение или удалить запись целиком.

Удаление записей дополнительных таблиц проходит «безболезненно», чего не скажешь о записях основной таблицы. В случае, когда запись основной таблицы связана с несколькими записями дополнительной таблицы, возможны два варианта поведения: не удалять основной записи, пока имеется хотя бы одна подчиненная запись (записи должен удалять пользователь), либо удалить основную запись и все подчиненные записи (каскадное удаление).

В многооконных системах (почти все современные программы) и, тем более, в распределенных информационных системах, работающих с базами данных, возникает проблема разрешения конфликтов между различными действиями над *одними и теми же объектами* (*совместного использования объектов БД*). Например, что делать в случае, когда один из пользователей локальной сети редактирует БД, а другой хочет изменить ее структуру? Для таких ситуаций в СУБД должны быть предусмотрены механизмы разрешения конфликтов.

Обычно при одновременной работе нескольких пользователей сети, а также работе нескольких приложений на одном компьютере или работе в нескольких окнах СУБД используются блокировки.

Блокировки могут действовать на различные объекты БД и на отдельные элементы объектов. Очевидной ситуацией блокировки объектов БД является случай одновременного использования объекта и попытки входа в режим разработки этого же объекта. Применительно к таблицам баз данных дополнительные блокировки могут возникать при работе с отдельными записями или полями.

Блокировки бывают явные и неявные. Явные блокировки накладываются пользователем или приложением с помощью команд. Неявные блокировки организует сама система, чтобы избежать возможных конфликтов. Например, в случае попытки изменения структуры БД во время редактирования информации устанавливается запрет реструктурирования БД до завершения редактирования данных. Более подробно типы блокировок в СУБД рассмотрены в подразделе 4.3.

Контрольные вопросы и задания

1. Назовите важнейшие задачи настройки и администрирования баз данных.
2. Охарактеризуйте подходы к выбору способа размещения файлов на диске.
3. Как определяется требуемый объем памяти на диске?
4. Сформулируйте соображения по распределению информации на дисках.

5. Охарактеризуйте назначение и технологию резервного копирования.
6. Назовите основные виды угроз в ВС.
7. Перечислите возможные последствия нарушения системы защиты.
8. Охарактеризуйте основные задачи защиты.
9. Укажите возможные уровни комплексной защиты информации в ВС.
10. Охарактеризуйте методы защиты ВС.
11. Дайте характеристику программно-аппаратных средств защиты.
12. В чем состоит смысл понятий идентификации и аутентификации пользователей?
13. Охарактеризуйте основные способы аутентификации.
14. Дайте характеристику алгоритмам шифрования DES и RSA.
15. Каково назначение систем защиты «Керберос» и «Кобра»?
16. Охарактеризуйте основные средства защиты информации в СУБД.
17. Назовите дополнительные средства защиты информации в БД.

Литература

1. Безруков Н. Н. Компьютерная вирусология: Справ. руководство. К.: УРЕ, 1991.
2. Беляев В. И. Безопасность в распределенных системах // Открытые системы, № 4, 1993. С. 36–39.
3. Бертолуччи Джейф. Быстрое полноэкранное видео на ПК // МИР ПК, № 5–6 (54), 1995. С. 34–36.
4. Бородаев В. А., Кустов В. Н. Банки и базы данных. Учебн. пособие. Л.: ВИКИ им. А. Ф. Можайского, 1989.
5. Воловик Е. М. Защита данных в распределенных системах // МИР ПК, № 10, 1995. С. 166–170.
6. Зима В. М., Молдовян А. А. Многоуровневая защита информационно-программного обеспечения вычислительных систем: Учебн. пособие. СПб.: Издательско-полиграфический центр СПбЭТУ, 1997.
7. Кирмайер М. Мультимедиа / Пер. с нем. СПб.: ВНВ-Санкт-Петербург, 1994.
8. Скальзо Б. Инструментарий для администраторов баз данных: мечты сбываются // PC WEEK / Russian Edition № 14 (138), 1998. С. 30–31.
9. Терлекчиев К. Kerberos на страже сети // Открытые системы, № 4, 1995. С. 40–43.
10. Четвериков В. Н., Ревунков Г. И., Самохвалов Э. Н. Базы и банки данных: Учебник для ВУЗов по специальности «АСУ». М.: Высшая школа, 1987.

9. Дополнительные вопросы применения баз данных

В разделе рассматривается характеристика аппаратно-программных платформ, выбор СУБД и структуры аппаратных средств, многопроцессорные системы обработки данных, перспективы развития СУБД и стандарты, используемые при разработке баз данных и информационных систем.

9.1. Программно-аппаратные платформы

Программно-аппаратные платформы, используемые при разработке и применении баз данных, оказывают существенное влияние на эффективность их функционирования. Проблемы выбора аппаратно-программных платформ для баз данных можно разделить на следующие составляющие:

- выбор СУБД;
- выбор аппаратных средств обработки баз данных.

Перспективным направлением повышения эффективности обработки данных является применение многопроцессорных систем обработки баз данных.

Выбор СУБД

Перед администратором БД, руководителем предприятия и обычным пользователем проблема выбора СУБД возникает чаще всего перед ее приобретением и при переходе на новые аппаратно-программные средства.

Подходы к выбору СУБД. Основным принципом выбора СУБД логично считать определение программного продукта, в наибольшей мере соответствующего предъявляемым требованиям. Практически решить эту задачу не очень просто. Во-первых, к СУБД предъявляется большое число требований и, главное, они с течением времени изменяются — по мере освоения системы требуются новые возможности. Во-вторых, СУБД имеют большое число параметров, что затрудняет их сравнение. Кроме того, информация о СУБД часто носит рекламный характер, не позволяющий сделать правильное суждение.

Рассмотрим технологию оценки характеристик СУБД и определения степени их соответствия предъявляемым требованиям. Выбор СУБД лучше всего производить с позиций лица, принимающего решение при неполной или противоречивой информации. Программные продукты обычно сопровождают следующая информация:

- сведения разработчиков и рекламная информация продавцов;

- информация конечных пользователей, разработчиков и администраторов, имеющих опыт работы с продуктом;
- информация аналитиков и экспертов.

При выборе продукта внимание следует сосредоточить на основных параметрах, а по остальным – проследить, чтобы не было «выпадения из области допустимости». Примером такого «выпадения» является невозможность работы с используемой ОС или отсутствие средств поддержки интерфейса ODBC.

Процедуру выбора СУБД удобно проводить в три этапа. Сначала на качественном уровне оценить предлагаемые программные продукты на предмет пригодности, сузив область выбора. Затем оценить технические характеристики отобранных систем более детально. И наконец, оценить производительность оставшихся продуктов для принятия окончательного решения.

К числу основных показателей пригодности программных продуктов можно отнести следующие:

1. Вид программного продукта.
2. Категории пользователей.
3. Удобство и простота использования.
4. Модель представления данных.
5. Качество средств разработки.
6. Качество средств защиты и контроля корректности базы данных.
7. Качество коммуникационных средств.
8. Фирма-разработчик.
9. Стоимость.

В конкретной организации имеется своя раскладка показателей на основные и дополнительные. Поиск нужного продукта рекомендуется начать с изучения потребностей и возможностей. Важно определиться, для чего нужен пакет: для разработки прикладных систем профессиональными программистами или для работы конечных пользователей в интерактивном режиме, что-то другое или несколько целей. Определяющими параметрами на первом этапе отбора являются вид программного продукта и категория пользователей.

К основным показателям в большинстве случаев относят первые два. Из оставшихся показателей, в зависимости от особенностей решаемых задач, в числе основных могут оказаться также четвертый, шестой и седьмой показатели. Рассмотрим перечисленные показатели.

Показатели пригодности. Виды СУБД и их классификация приведены в подразделе 1.3, рассмотрим остальные показатели пригодности СУБД.

Категории пользователей. Программный продукт, относящийся к классу СУБД, в общем случае, может быть предназначен для следующих категорий пользователей:

- профессиональных программистов – разработчиков СУБД, серверов БД и других программ;

- администраторов БД;
- квалифицированных пользователей, разрабатывающих приложения;
- конечных (неквалифицированных) пользователей;
- различных комбинаций перечисленных категорий.

При выборе программных продуктов следует отдавать предпочтение программам более широкого назначения. Не случайно многие популярные полнофункциональные СУБД имеют средства как для пользователей и администраторов, так и для разработчиков. Так, СУБД Microsoft Access позволяет для программирования приложений использовать Visual Basic для приложений.

Удобство и простота использования. Понятие удобства и простоты использования довольно расплывчатое, со временем изменяется и, кроме того, ужесточается с точки зрения предъявляемых требований. Удобство и простоту использования программ качественно характеризует следующее:

- понятные процедуры установки программных продуктов (особенно сетевые установки с множеством рабочих мест);
- удобный и унифицированный интерфейс конечного пользователя;
- простота выполнения обычных операций: создания БД, навигации, модификации данных, подготовки и выполнения запросов и отчетов и ряда других;
- наличие интеллектуальных подсистем подсказок, помощи в процессе работы и обучения, включая примеры.

Модель представления данных. В настоящее время наиболее распространенной и отработанной теоретически и практически является реляционная модель данных (раздел 2). Перспективными являются модели с объектной ориентацией, поскольку они обладают большими возможностями отражения семантики предметной области. Поэтому в большинстве случаев предпочтение отдают системам с реляционной и объектно-ориентированной моделью данных. Специфические задачи, разумеется, могут диктовать необходимость использования других моделей представления данных.

Качество средств разработки. При оценке качества средств разработки учитывается следующее: возможности создания пользовательских интерфейсов; мощность языка создания программ (автоматическая генерация кода, отладка, обеспечение целостности данных на уровне процессора БД, а не с помощью команд языка); автоматизация разработки различных объектов: экранных форм, отчетов, запросов. Предпочтение отдается системам, имеющим полнофункциональные генераторы (Мастера, Построители и т. п.) и обеспечивающим удобство работы пользователя.

Качество средств защиты и контроля корректности базы данных. Актуальное требование защиты информации в современных информационных системах требует принятия адекватных мер в СУБД. Доступ к функциям защиты должен предусматриваться на уровне средств разработки программ

и на уровне пользователя (обычного пользователя, администратора БД) (см. подраздел 8.2).

К важнейшим функциям контроля корректности БД относятся следующие:

- обеспечение уникальности записей БД по первичному ключу (не каждая полнофункциональная СУБД это делает);
- автоматический контроль целостности связей (ссылочная целостность) между таблицами во время выполнения операций обновления, вставки и удаления записей (подраздел 3.4);
- проверка корректности значений в БД (контроль типа данных, совпадение с шаблоном, определение диапазона допустимых значений, контроль значения по справочной таблице и др.).

Качество коммуникационных средств. При оценке качества коммуникационных средств обращают внимание на следующие свойства программных продуктов:

- поддержку сетевых протоколов, обеспечивающих работу продукта в различных сетях;
- поддержку стандартных интерфейсов с БД: SQL, ODBC, IDAPI, SAA и др. (подраздел 9.3);
- наличие средств групповой работы с информацией БД (языковые средства разработки; функции интерфейса пользователя; функции администратора БД по организации групп, разграничению полномочий, защите от несанкционированного доступа и т. д.);
- способность использовать и модифицировать БД других форматов без импортирования или преобразования.

Фирма-разработчик. При отборе программных продуктов немаловажное значение имеет авторство продукта. Солидность фирмы-разработчика пакета, как правило, дает следующие преимущества:

- высокое качество продукта;
- наличие документации и методических материалов;
- наличие «горячей линии» для консультаций по возникающим проблемам;
- высокую уверенность в появлении более совершенной версии.

Заметим, что очередные версии СУБД в среднем появляются достаточно быстро. При выборе продукта следует обратить внимание на дату его появления. Возможно, что в данный момент на подходе очередная версия фирмы-конкурента, которая по многим параметрам лучше рассматриваемой. В дальнейшем ситуация может измениться в обратную сторону.

Следует отдавать предпочтение фирмам с твердым финансовым положением и перспективной динамикой развития аппаратно-программных средств. В качестве показателей «благополучия» можно использовать годовой оборот, численность состава, объем продаж вообще и интересующего продукта в частности и т. д.

Стоимость. На стоимость программных продуктов в основном влияют вид программного продукта и фирма-разработчик. Стоимость полно-

функциональных СУБД обычно колеблется в пределах \$500–\$1000. Намного дороже серверы БД, цена их ядра процессора БД колеблется от нескольких сот долларов до пятисот тысяч долларов. Общая стоимость включает в себя также стоимость прикладного инструментария, средств настройки конфигурации системы, администрирования БД и сопровождения. Иногда общая стоимость крупных систем, построенных на базе реляционных БД, достигает миллиона долларов. Основным фактором, определяющим общую стоимость системы, чаще всего является число поддерживающих пользователей.

С появлением сети Интернет стало возможно бесплатно приобретать программные продукты, в том числе СУБД. Примером такого продукта является свободно распространяемая и основанная на модели «клиент-сервер» постреляционная СУБД POSTGRES95.

Представителями бесплатных СУБД с открытым кодом также являются программы MySQL, PosgreSQL и Firebird. Такие системы очень популярны в среде малого и среднего бизнеса, обычно поддерживают только основные функции СУБД (хотя перечень этих функций от версии к версии расширяется), могут функционировать на различных платформах. Так, широко используемая СУБД MySQL может работать в среде операционных систем Linux, Windows, Solaris, Mac OS X, FreeBSD, HP-UX, IBM AIX 5L и других.

Технические характеристики. Разнообразие СУБД на уровне технических характеристик еще больше, чем на качественном уровне. Остановимся на наиболее существенных из них. Для удобства представления характеристик сведем их в таблицу.

Таблица 9.1

Основные технические характеристики СУБД

Вид характеристики	Характеристика
Общие параметры	Операционная среда (типы поддерживаемых ОС и коммуникационных протоколов) Потребность в оперативной памяти Ограничение на максимальный объем БД Ограничение на количество одновременных подключений (пользователей, приложений)
Ограничения на операции над данными	Максимальный размер колонки (поля) Максимальный размер строки, кбайт Максимальное число полей в таблице Максимальное число индексных полей Максимальное число строк в таблице Максимальное число одновременно открытых таблиц

Типы данных	Текстовый постоянной длины Текстовый переменной длины Числовой Целочисленный Десятичный с фиксированным числом знаков после запятой Десятичный с плавающей точкой Дата Время Дато-временной (данные о датах и/или времени) Логический Комментарии Виртуальный (вычисляемый) Двоичный – хранение графической, аудио-, видео- и другой информации (OLE, BLOB и т. д.) Гиперссылка (hyperlink) – ссылка на файлы или документы, находящиеся вне базы данных на локальном компьютере или в сети
Возможности средств формулировки и выполнения запросов	Вид языка запросов: SQL, QBE свой собственный Вид интерфейса запросов: командная строка, шаблон (стандартная форма) Прекомпилятор и оптимизатор Сохранение запросов Ограничение на число таблиц и виды связей Максимальное число полей для поиска Сортировка: по одному любому полю, по нескольким полям Наличие вычислений в запросах Групповые операции и операции над множествами
Работа в много-пользовательских средах	Типы блокировок: исключительные, общие Уровни блокировок: блокировка БД, блокировка объектов разработки (текстов программ, отчетов, экранных форм и пр.), блокировка таблицы (файла), блокировка записи, блокировка поля
Работа в много-пользовательских средах	Идентификация станции, установившей блокировку Обновление информации после модификации Контроль за временем и повторные обращения Обработка транзакций
Инструментальные средства разработки приложений	Генератор интерфейсов пользователя (в т. ч. экранных форм, кнопок, меню, окон и т. д.) Генератор отчетов Генератор приложений Версия времени выполнения (ядра для запуска программ) Генерация независимых exe-модулей
Импорт и экспорт	ASCII-файлы DBF-формат WK-формат XLS-формат Другие форматы

Многие из характеристик систем достаточно очевидны и в комментариях не нуждаются. Заметим, что не для всех видов СУБД имеют место приведенные характеристики.

Оценка производительности. Анализом и испытанием СУБД занимаются различные организации, в частности лаборатория журнала PC Magazine. Предлагаемая ею методика анализа производительности СУБД применима для исследования программ класса СУБД, коротко ее рассмотрим.

Тестирование реляционных СУБД проводится с помощью эталонных тестов из набора AS³AP (ANSI SQL Standard Scalable and Portable). В них контролируется широкий спектр часто встречающихся операций с БД и моделируются однопользовательская и многопользовательская среды. Испытываемая БД состоит из четырех таблиц по 100 тысяч записей.

Основные виды применяемых тестов носят названия: «Выборка», «Полное сканирование», «Загрузка и индексация», «Обновление», «Чтение с произвольной выборкой», «Запись с произвольным доступом» и «Генерация отчета», указывающие суть этих тестов. Для проведения тестирования каждой фирмой-поставщиком программного продукта разработаны соответствующие прикладные программы, инициирующие тестовые испытания.

В teste «Выборка» измеряется, насколько быстро каждый программный продукт может выполнять однотабличный запрос, ответ на который содержит определенную процентную долю строк таблицы. Запросы включают числовые и текстовые данные. Используются индексированные поля, а значения должны попадать в заданный диапазон.

Тест «Полное сканирование» измеряет время, необходимое для поиска значения в неиндексированном поле, которое отсутствует в таблице. Тем самым моделируется самый неблагоприятный случай запроса к БД.

В teste «Загрузка и индексация» проводится оценивание того, как быстро программы могут импортировать исходную БД и создать определенное число индексов.

Тест «Обновление» предназначен для измерения времени обновления индексов при выполнении операций модификации (Modify), вставки или добавления (Append) и удаления (Delete) 1000 записей.

Тест «Чтение с произвольной выборкой» позволяет определить максимальное число параллельных обращений к данным, которые способен обработать испытуемый пакет. Для получения этой характеристики, на рабочих станциях генерировались запросы со случайными номерами записей одной таблицы, которые должен был выполнять пакет, находящийся на отдельном компьютере. Принимаемые ответы на запросы на рабочих станциях уничтожались.

Тест «Запись с произвольным доступом» похож на предыдущий тест, с той разницей, что каждая из рабочих станций случайно выбирает обновляемую запись, в которой изменяется целочисленное поле.

Тест «Генерация отчета» предполагает задействование одной рабочей станции для подготовки и печати собственно отчета, а восьми других машин – для имитации потока требований к основной станции при групповой работе. Имитация проводится с помощью теста «Чтение с произвольной выборкой».

По параметрам производительности оценивались 10 наиболее высококачественных на тот период времени СУБД: Clarion Database Developer 3.0, DataEase for DOS 4.53, DataFlex 3.05 (DOS), dBase IV 2.0, Microsoft Access 2.0, Microsoft FoxPro 2.6 for DOS (Professional Edition), Microsoft FoxPro 2.6 for Windows (Professional Edition), Paradox for DOS 4.5, Paradox for Windows 4.5 (Professional Edition) и R:BASE 4.5 Plus. В целом, лидирующее положение заняли обе версии пакета Microsoft FoxPro. Этот продукт, к сожалению, практически не содержит встроенных средств контроля целостности БД.

Отдавать предпочтение следует продуктам с развитыми средствами контроля целостности и корректности данных, широкими возможностями пользовательского интерфейса при неплохих характеристиках производительности. К таким изделиям из числа названных относятся Paradox и Access.

Замечание.

Известна также другая группа тестов, разработанных Советом по Производительности Обработки Транзакций (TPC – Transaction Processing Performance Council). В набор тестов TPC, который постоянно совершенствуется и дополняется, входят тесты: TPC Benchmark A (кратко – TPC-A), TPC-B, TPC-C, TPC-D и TPC-E. Эти тесты созданы для определения производительности и соотношения цена/производительность аппаратно-программных систем на задачах оперативной обработки транзакций (OLTP – On-Line Transaction Processing).

Остановимся на трех наиболее отработанных и распространенных тестах: TPC-A, TPC-B и TPC-C.

Тест TPC-A предполагает тестирование с помощью приложения всех программно-аппаратных компонентов системы, включая пользовательские терминалы и сеть передачи данных. В процессе тестирования моделируется сеть банковских служащих, принимающих депозиты и осуществляющих выдачу вкладов.

Оцениваемое соотношение цена/производительность вычисляется как частное от деления стоимости системы на максимальное количество зафиксированных транзакций в секунду. В общую стоимость системы включается стоимость аппаратуры, программного обеспечения, а также стоимость обслуживания аппаратной части в течение пяти лет.

Назначение **теста TPC-B** состоит в определении совместной производительности СУБД и аппаратной платформы, состоящей из вычислительной системы и дисковой памяти, с помощью того же приложения, что и в teste TPC-A. По сравнению с предыдущим вариантом, из схемы тестирования исключены этапы считывания и вывода информации на терминал. В результа-

те тестирования получают данные о производительности (количество зафиксированных транзакций в секунду), а также соотношении цена/производительность, вычисляемом аналогично тесту TPC-A.

В *тесте TPC-C* проводится многостороннее исследование систем в более сложных условиях оперативной обработки информации. На вход системы подается смесь транзакций, осуществляющих чтение и интенсивное обновление БД. Здесь используются многочисленные таблицы разных размеров, структуры связей с другими таблицами. Поступать транзакции могут одновременно разных типов (короткие и длинные, в зависимости от трудоемкости обработки), а выполняться – в оперативном и в отложенном режиме. Тестирование производится на примере моделирования деятельности типичного склада.

Выбор структуры аппаратных средств

При создании информационно-вычислительных систем, предполагающих обработку данных в базах, кроме выбора СУБД требуется решать проблему обоснования структуры программных и аппаратных средств. Для определения рациональной структуры программно-аппаратного обеспечения обработки данных в общем случае приходится отвечать на следующие вопросы:

- какую структуру (однопотоковую или многопотоковую) имеет СУБД;
- применяются ли мониторы транзакций;
- можно ли использовать архитектуру клиент-сервер;
- сколько одновременно активных пользователей должна поддерживать система;
- можно ли из всего множества запросов выделить основной – шаблон (образец);
- какова стратегия индексации;
- какие запросы нужно поддержать индексацией, а какие можно реализовать с помощью сканирования базы данных;
- каков чистый размер базы данных;
- достаточно ли дисковых накопителей и интерфейсных адаптеров для обеспечения обработки предполагаемой нагрузки;
- имеются ли отдельные диски для журналов СУБД и архивов;
- достаточно ли емкость дисковой памяти для хранения данных, индексов, временных таблиц и для возможного увеличения объема данных;
- достаточно ли процессоров для работы с предполагаемым числом пользователей;
- требуется ли выделенная сеть для организации связи между клиентскими системами и сервером в системах клиент-сервер;
- согласована ли стратегия резервного копирования с типом, числом и местом размещения устройств резервного копирования.

Для получения ответа на перечисленные и подобные вопросы нужно прежде всего учитывать следующие факторы:

- 1) особенности прикладной задачи, а также методы и средства ее решения;
- 2) характеристики выбранной СУБД;
- 3) возможности и эффективность функционирования операционной системы;
- 4) характеристики аппаратной части и сетевого оборудования.

Замечание.

Неэффективное решение прикладной задачи может свести на нет остальные усилия. Например, для выбора записи можно использовать сканирование всей таблицы с проверкой условия или операцию выборки – оператор SELECT языка SQL. В первом случае в среднем требуется просмотреть половину таблицы, что при размере таблицы в десятки гигабайтов может составлять единицы-десятки минут. Во втором случае операция выборки составляет не более нескольких секунд. Другой пример: игнорирование при создании базы данных преимуществ определения ключей и индексирования таблиц. Особенный выигрыш от последнего в производительности получается при выполнении типовых или повторяющихся запросов к базе данных. Главное, что реорганизация базы данных здесь не требует изменения кода приложения. А эффект может быть существенным. Неучет этого обстоятельства сильно замедляет работу приложения.

Параметры прикладной задачи тесно связаны с характеристиками выбранной СУБД (*вторая составляющая*). Эта связь особенно сильно проявляется в случае, когда приложение выполняется в режиме интерпретации СУБД (подраздел 1.4). Детальный перечень характеристик СУБД приведен в табл. 9.1 настоящего подраздела. При установке СУБД и при ее эксплуатации следует не забывать о сборе и анализе статистики о процессах и событиях. Нужно обратить внимание на такие явления, как обработка транзакций (время выполнения), возникновение блокировок при использовании разделяемых ресурсов, использование кэш-памяти при обращении к дискам, а также нехватка буферной или дисковой памяти. Все негативные явления, как правило, существенно снижают показатели работы информационной системы и в то же время могут быть своевременно выявлены и устранены настройкой СУБД, приложения и операционной системы.

Программной средой функционирования приложения и/или СУБД является операционная система (*третья составляющая*). От нее тоже зависит интегральная оценка работы информационной системы. Иногда выбор ОС носит принципиальный характер. Например, требование со стороны СУБД среди Windows, отвергает возможность использования UNIX и MS DOS.

Важнейшей частью вычислительной системы является аппаратура (*четвертая составляющая*). Поскольку необходимым условием построения эффективной информационной системы является обоснованное определение

структуры аппаратной части системы, остановимся на этой задаче более подробно.

При решении задач определения структуры программно-аппаратных средств автоматизированной системы обработки данных в большинстве случаев в силу большой неопределенности и огромного числа противоречивых факторов для принятия решения достаточно экспертной оценки.

Предположим, мы имеем клиент-серверную систему с выбранными и рационально сконфигурированными ОС и СУБД, а также эффективными приложениями. Стоит вопрос о том, какими возможностями должны обладать компьютеры этой распределенной системы с учетом имеющихся финансовых ограничений.

Вопрос распределения вычислительной мощности между клиентскими и серверными машинами, очевидно, следует решать под углом зрения используемой модели клиент-сервер (подраздел 4.2). Более нагруженная часть системы должна быть обеспечена большими возможностями. При этом нужно учитывать специфику решаемых задач на клиентской и на серверной сторонах.

Ввиду большого разнообразия решаемых задач относительно **клиентской части** ограничимся общей рекомендацией: проанализировать потребности ресурсов ЭВМ и в пределах имеющихся финансовых их удовлетворить. Например, приложения, требующие обработки видеоизображений, должны выполняться на компьютерах с хорошими характеристиками видеосистем (достаточным объемом видеопамяти, неплохим разрешением мониторов, обеспечивающим нужную скорость обработки изображений быстрым действием процессора).

Относительно **серверной части** систем можно высказаться более определенно, поскольку его требования к системным ресурсам в меньшей степени зависят от специфики решаемых прикладных задач. Рассмотрим специфику использования серверами важнейших ресурсов компьютера: основной памяти, центрального процессора и внешней памяти.

Основная память. Характеристики основной памяти (ОП), главным образом ее объем, являются важнейшими среди других характеристик компьютера. При выборе компьютера целесообразно предусмотреть возможность увеличения объема памяти в будущем. Многие современные компьютеры позволяют легко нарастить первоначально имеющийся объем памяти.

В основной памяти хранятся программы и данные. При работе с СУБД, как правило, большая часть ОП используется в качестве буфера (кэша) для обмена с внешней памятью. Наличие кэша достаточного объема позволяет уменьшить число операций физического ввода/вывода и повышает производительность вычислительной системы. Для оценки получаемого выигрыша отметим, что обращение к основной памяти выполняется примерно в 30000 раз быстрее, чем к магнитному диску.

Основную долю в общем объеме ОП занимает область кэша, рассмотрим как определить его размер. Точное определение оптимального размера буферной области – непростая задача. Главная причина этого – в заранее неизвестных потребностях в данных со стороны приложений, зависящих от множества факторов (решаемых задач, характеристик клиентских машин, категорий пользователей и т. д.).

Для определения размера кэша можно воспользоваться следующими подходами: моделированием процессов обработки информации на сервере или подбором параметров кэша на основе анализа статистики потребления памяти. Многие современные СУБД имеют средства оценки эффективности использования кэша. Часто кэш реализуется как массив разделяемой памяти, размер которого определяется специальным параметром в управляющем файле или таблицах СУБД. При сложности использования названных подходов, а также при первичной установке сервера для определения размера кэша можно воспользоваться одним из следующих эмпирических правил.

A. Правило «пяти минут». Соотношение цен основной и дисковой памяти таково, что экономически целесообразно кэшировать данные, обращения к которым происходят более одного раза каждые пять минут. Для определения размера кэша нужно сначала на уровне всей системы сложить объемы всех данных, которые приложения предполагают использовать более часто, чем один раз в пять минут. Дополнительно следует зарезервировать еще 5–10% от суммарного объема данных для хранения индексов, хранимых процедур и другой управляющей информации СУБД. Правило широко используется при конфигурировании серверов баз данных, для которых точное определение размера кэша затруднительно.

B. Правило «90/10». Исследования современных БД показывают, что 90% всех обращений выполняются к 10% данных («горячие» данные). Более того, обращения к «горячим» данным, в свою очередь, тоже подчиняются правилу «90/10». Таким образом, около 80% всех обращений к данным связаны с доступом к примерно 1% данных. Отсюда следует, что для кэша требуется объем основной памяти не менее 1% от «чистого» объема хранимых данных, не считая памяти под индексы и другую служебную информацию.

Правило применимо для больших баз данных, когда 10% общего объема базы данных принципиально невозможно или дорого разместить в кэше. Так, для скромных по сегодняшним меркам баз данных объемом 5 Гбайтов 10% составляет 500 Мбайтов. Сконфигурировать машину класса Pentium с таким объемом кэша пока проблематично и дорого. Обеспечить кэш объемом в 1% всех данных и для очень больших баз данных – вполне реально.

C. По числу пользователей. Грубой оценкой требуемого объема основной памяти под кэш является величина, получающаяся при выделении от 50 до 300 кбайтов на каждого пользователя. Правило целесообразно применять для

многопользовательских систем, когда невозможно определить размер кэша другим способом.

Замечания.

- Чрезмерное увеличение размера кэша обычно не дает существенного эффекта. Более того, эффект может оказаться обратным: система станет работать медленнее. Причиной этого может быть лишение пользовательского приложения или сервера СУБД требуемой для нормальной работы основной памяти.
- Выделение памяти под кэш может уменьшить размер виртуальной памяти, используемой операционной системой для буферизации операций файловой системы.
- В системе должно обеспечиваться пространство для традиционного использования памяти. Так, в вычислительных системах под управлением UNIX желательно иметь объем основной памяти не менее 16 Мбайтов для операционной системы, 2–4 Мбайта – для ряда программ СУБД и достаточно места для размещения в памяти двоичных кодов приложения. Объем двоичных кодов приложения составляет обычно 1–2 Мбайта, но иногда они могут достигать десятков Мбайтов. Поскольку операционная система обеспечивает разделение памяти между множеством процессов, достаточно зарезервировать пространство для одного приложения.
- Для современных серверных систем существует также следующее эмпирическое правило: неразумно комплектовать машину памятью менее чем примерно по 64 Мбайтов на процессор. В противном случае в системе возникает интенсивная фрагментация памяти.

Центральный процессор. Потребление процессорных ресурсов в серверной системе сильно зависит от используемого приложения, СУБД, индивидуальных пользователей и от времени дня. Поэтому однозначно дать рекомендации по составу и мощности центрального процессора (ЦП) в общем случае невозможно. Некоторые оценки по способностям используемых процессоров можно дать по результатам тестирования.

Так, результаты теста ТРС-А для восьмипроцессорного SPARCserver 1000 показывают, что он способен обрабатывать запросы от 4000 пользователей, или от 500 пользователей на процессор. Этот результат достигнут путем тщательной настройки ОС Solaris, СУБД Oracle и самого теста. Для большинства пользователей более реалистической верхней границей числа пользователей на процессор для транзакций типа ТРС-А возможно составит порядка 100–200 пользователей на один процессор 50 МГц SuperSPARC.

Дисковая система ввода/вывода. Как известно, любое обновление базы данных связано с обновлением индексов и с выполнением записи в журнал. Каждая из этих операций может вызывать последовательное и/или произвольное обращение к дисковой памяти. Примером последовательного обра-

щения к дисковой памяти является операция сканирования (последовательного просмотра) или частичного сканирования индексов (отсортированных) перед обращением к записям индексированной БД.

Без хорошего знания приложения, стратегии индексации, принятой администратором базы данных, а также механизмов поиска и хранения СУБД в общем случае невозможно точно сказать какие преимущественно выполняются операции ввода/вывода. Поэтому обычно предполагают, что доминирует операция произвольного доступа к диску. Эта операция вызывает большую нагрузку на дисковую систему ввода/вывода, нежели последовательный доступ.

При подборе дисковых накопителей, кроме размера общей емкости дисковой памяти, следует помнить о достаточной пропускной способности дисковой подсистемы. Почти всегда лучшие характеристики производительности достигаются при меньшей емкости диска, даже когда больший диск имеет превосходные характеристики по всем параметрам.

На производительность дисковой подсистемы ввода/вывода большое влияние оказывает количество и вид используемых системных шин. Различные шины поддерживают работу разного количества дисков. Так, на однойшине Fast SCSI-2 (10 Мб/с) можно сконфигурировать небольшое число дисков (3–5), а нашине Fast-and-Wide SCSI (20 Мб/с) – до 8–10 дисков.

Помимо характеристик емкости МД и пропускной способности шин, на производительность подсистемы ввода/вывода большое влияние оказывает уровень загруженности этих компонентов. Практика показывает, что в случае пиковых нагрузок степень загруженности шины должна поддерживаться на уровне 40%, а степень загрузки дисков – на уровне 60%.

Многопроцессорные системы обработки баз данных

Одним из путей повышения производительности и живучести систем обработки информации в базах данных является применение вычислительных систем с улучшенной архитектурой. Для этого проводят исследования и разработку новых методов и алгоритмов по следующим направлениям:

- хранение и обработка информации в оперативной памяти компьютера;
- выполнение запросов в многопроцессорных и многомашинных вычислительных системах с различными архитектурами;
- эффективная реализация типовых операций в БД (сортировка, поиск, реорганизация, пакетная загрузка/выгрузка и пр.);
- восстановление данных в случае отказа узлов вычислительной системы и т. д.

Одними из основных параметров систем, использующих БД, являются объемные (максимальный объем данных и возможность их распределенного хранения) и временные (время выполнения отдельной операции, транзакции, или всей работы). Значения этих параметров существенно зависят от мощности и организации вычислительной среды функционирования СУБД.

Фирмы, разрабатывающие СУБД, для улучшения названных характеристик использовали следующие варианты аппаратных средств:

1. Традиционные однопроцессорные ЭВМ повышенной производительности.
2. Специализированные процессоры баз данных – машины баз данных.
3. Вычислительные системы на базе многопроцессорных структур.

Несмотря на широкое распространение систем первого типа, в настоящее время лучшие результаты показывают системы третьего типа. Примерами параллельных систем баз данных являются Teradata, Tandem, Gamma, Bubba и Arbre. Рассмотрим более подробно многопроцессорные вычислительные системы, предварительно дав определение эффективности.

Под **эффективностью** будем понимать свойство, характеризующее качество реализации системой поставленных перед ней целей функционирования и применения.

Эффективной может оказаться несложная информационная система, реализованная на основе специализированной вычислительной системы с низкими характеристиками быстродействия и емкости оперативной и внешней памяти. С другой стороны, есть много новых задач, требующих следующего:

- высокой производительности вычислительной системы;
- значительного объема БД в сотни мегабайтов и терабайтов;
- решения в реальном масштабе времени;
- одновременной обработки разнородных подзадач (массовый ввод и модификация данных, поддержка принятия решения, пакетная обработка);
- одновременного обслуживания большого числа запросов.

Анализ перечисленных требований показывает, что обеспечить их, ограничившись универсальной однопроцессорной вычислительной системой или специализированным средством с аппаратной реализацией часто выполняемых операций, весьма сложно. Примером уникальной вычислительной системы, которая в некоторой степени могла бы удовлетворить предъявляемым требованиям, можно считать суперкомпьютер баз данных (Super Database Computer – SDC) токийского университета. В нем используется комбинированный аппаратно-программный подход к решению проблемы производительности. Основное обрабатывающее устройство из одного или нескольких процессоров с совместно используемой памятью дополняется специализированным устройством сортировки и дисковой подсистемой.

Более перспективными с точки зрения удовлетворения перечисленным выше требованиям оказываются многопроцессорные вычислительные системы.

Определилось два основных архитектурных направления многопроцессорных систем: *сильносвязанные* и *слабосвязанные* вычислительные системы.

К *сильносвязанным* вычислительным системам, или системам с *разделением ресурсов*, относятся следующие:

- системы с совместно используемой (разделяемой) памятью (рис. 9.1а), в которых процессоры имеют доступ к общей оперативной памяти и ко всем дискам (IBM/370, Digital VAX, Sequent Symmetry);
- системы с совместно используемыми дисками (рис. 9.1б), в которых каждый процессор имеет свою основную память и обеспечивается прямым доступом ко всем дискам (IBM Sysplex и первоначальная версия Digital VAXcluster);
- системы с массовым параллелизмом – системы с сотнями и тысячами процессоров, произвольным образом объединяемых друг с другом (рис. 9.1в).

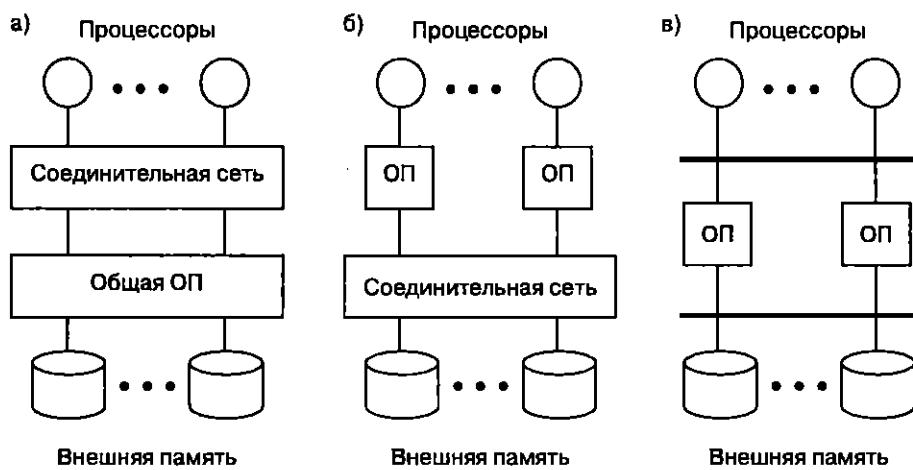


Рис. 9.1. Сильно связанные вычислительные системы

Слабосвязанные многопроцессорные вычислительные системы, или системы без совместного использования ресурсов, представляют собой совокупность компьютеров, объединенных в единую систему быстродействующей средой передачи информации (рис. 9.2). Процессоры поддерживают связь друг с другом путем передачи сообщений. Примерами слабосвязанных многопроцессорных систем являются: система Teradata, которая может иметь свыше 1000 процессоров и тысячи дисков, и система Gamma, работающая на Intel iPSC/2 Hypercube с 32 узлами, каждый из которых имеет собственный диск.

Для названных задач из указанных классов параллельных систем предпочтительными чаще оказываются **слабосвязанные** системы.

Во-первых, в системах с разделением ресурсов требуется сложная операционная система (использующая часть ресурсов), отслеживающая и разрешающая конфликты из-за обращения к совместно используемым ресурсам. Кроме того, в них при добавлении нового процессора замедляется работа

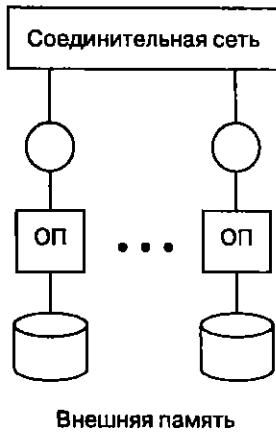


Рис. 9.2. Слабосвязанная вычислительная система

остальных процессоров. В однопроцессорных системах основной причиной снижения производительности для многозадачного и многопользовательского режимов работ с базами данных являются операции загрузки и выгрузки кэш-памяти.

Во-вторых, системы с массовым параллелизмом, по-видимому, ожидает большое будущее, но в настоящее время они не имеют массового применения из-за высокой стоимости компонентов компьютеров и сложной организации вычислительного процесса.

Основным достоинством слабосвязанных вычислительных систем является легкость наращивания числа процессоров до сотен или даже тысяч без существенных помех в их работе. В системах с разделением памяти максимальное число процессоров пока составляет 32. Системы без совместного

использования ресурсов позволяют достичь почти линейного ускорения и расширяемости при обслуживании сложных реляционных запросов и транзакций.

Еще одним достоинством слабосвязанных вычислительных систем является высокая надежность и простота управления процессом обработки информации. Кроме того, подсистемы связи в них не должны иметь высокую производительность, как в системах с совместным использованием ресурсов.

Достичь высоких временных показателей обработки данных в слабосвязанных вычислительных системах удается благодаря использованию реляционной модели. Реляционный запрос хорошо подходит к параллельному выполнению: из операторов над отношениями можно составить параллельный граф потоков данных.

Основными методами *распараллеливания обработки* данных являются: конвейеризация и разнесение обработки. Конвейеризация состоит в выделении стадий выполнения операций над данными базы и распределении отдельных стадий по обрабатывающим узлам вычислительной системы. Разнесенный параллелизм возможен в случаях, когда допускается разделение (разнесение) источников данных и независимая их обработка.

В параллельных реляционных СУБД используются оба вида распараллеливания (часто одновременно), причем существенно больший эффект может дать второй из них.

Факторами, ограничивающими использование конвейеризации, являются: незначительное число этапов подавляющего числа реляционных операторов, а также эффект «перекоса», состоящий в различной трудоемкости этапов выполняемых операций. Выигрыш от разделения данных также

может быть различным – в зависимости от правильности выбора используемых алгоритмов и методов (кольцевое разделение, с хэшированием, на основе диапазона значений, на основе частоты обращения к кортежам и т. д.).

В реализации параллельных систем баз данных имеются следующие нерешенные проблемы:

- обеспечение высоких временных характеристик при смешанной нагрузке;
- оптимизация параллельных запросов;
- выбор оптимальных методов физического проектирования баз данных (разделение данных, выбор индексов для таблиц и т. д.);
- разработка методов и средств реорганизации данных в режиме on-line;
- исследование алгоритмов конвейеризации и т. д.

При организации распределенных информационных систем применяют более совершенные схемы обеспечения целостности совокупности БД. Так, вместо двухфазной фиксации транзакций (когда во всех узлах системы синхронно фиксируется начальное и конечное состояния всех БД), применяется, например, асинхронное тиражирование данных (подраздел 4.3).

9.2. Перспективы развития СУБД

Анализ современных СУБД и реализованных на их основе приложений позволяет предположить следующие направления их развития:

- 1) поиск более совершенных моделей представления и типов данных в базах;
- 2) разработка новых архитектур СУБД;
- 3) расширение областей применения БД;
- 4) улучшение сервиса конечных пользователей, администраторов и разработчиков.

В рамках **первого направления** представляют интерес СУБД, поддерживающие несколько моделей или одну интегрированную модель и позволяющие удобно программировать вычисления, обрабатывать символьную и графическую информацию, работать со знаниями, аудио- и видеоинформацией, осуществлять доступ к распределенной информации, организовывать телеконференции, обучение и выполнять другие операции. На пути к решению этой проблемы находится попытка поддержки во многих современных СУБД различных типов двоичных данных и типа **гиперссылка**.

Новые архитектуры СУБД. Современные информационные системы в ряде случаев требуют от СУБД возможности хранить и обрабатывать данные объемом порядка Петабайтов (Petabyte – 10^{15} байтов). Несмотря на значительный рост возможностей по объему магнитных и магнитоопти-

ческих дисков, вряд ли их будет достаточно для информационных систем, работающих со сверхбольшими объемами данных. В связи с этим говорят о необходимости организации нового уровня иерархии носителей – «третичной памяти». Устройствами третичной памяти могут быть устройства в виде стоек магнитных дисков или лент с автоматически сменяемыми носителями.

Примером такого устройства является буферная система VSM (Virtual Storage Manager – менеджер виртуальной памяти) корпорации StorageTek. Система VSM накапливает данные и сохраняет их на жестких дисках в буфере данных, где они складируются в виде виртуальных томов на магнитных лентах (до 100 000 виртуальных томов на каждом дисковом буфере). Максимальная скорость передачи данных пользователя – до 45 Мбайтов/с.

Еще одним примером является система CD Storage System корпорации Compaq Computer. Она предоставляет сетевым пользователям доступ к данным на компакт-дисках, размещенных в корпусе фирмы Micro Design International. Всего может быть до семи блоков (корпусов), каждый из которых вмещает по семь приводов компакт-дисков. Объем памяти каждого блока – до 4,5 Гбайтов. Можно установить до 49 дисководов на сервер и управлять ими как одним логическим диском.

К **новым областям применения** СУБД можно отнести следующие два класса задач: обработки сверхбольших объемов информации и распределенной обработки информации в сетях ЭВМ.

Примером системы, решающей одну из задач первого класса, является проектируемая информационная система наблюдения Земли EOS (Earth Observing System), основным элементом которой является база данных EOSDIS (EOS Data and Information System – система данных и информации).

Примерами задач второго типа являются задачи поиска и отбора информации в Интернете, организации коллективного проектирования в территориально разнесенных организациях, обмена материальными, информационными, денежными и другими ресурсами с электронным оформлением.

Если брать **качество сервиса** в широком смысле, то перспективные СУБД позволяют решать стоящие прикладные задачи с лучшим качеством. Для этого они будут опираться на более совершенную элементную базу (повышение объема хранимых данных, увеличение производительности обработки запросов), иметь более совершенную программную организацию (распределенная обработка, безопасность хранимой информации, защита прав собственности), обладать более гибкими и удобными интерфейсами для программистов, пользователей и администраторов БД.

Осознание необходимости хранения в базах данных не только информации о предметной области, но и информации разработчиков приложений привела к тому, что некоторые крупные фирмы (Oracle, Microsoft) заявили о

скором появлении программных продуктов управления метаданными – объектно-ориентированных репозиториях. Такие репозитории полезны администраторам БД в управлении метаданными, а также разработчикам, так как позволяют при разработке приложений многократно использовать готовые компоненты.

Одним из новых требований в области информационных технологий является обеспечение безостановочной работы. С одной стороны, возможности компьютеров, а с другой – конкуренция привели к тому, что некоторые информационные системы работают непрерывно. Появился так называемый «стандарт готовности», который определяется как возможность пользователя совершать интерактивное обновление данных 24 часа в сутки, 7 дней в неделю, 52 недели в год. Часто безостановочный режим работы называют «7*24-» или «24*365-работой». Это означает, что в каждый день года в любое время пользователю доступны информационные ресурсы, без скидок на выходные и праздничные дни.

Такой уровень качества прикладных систем, использующих БД, помимо мер повышения надежности и устойчивости непрерывной работы, выдвигает и новые требования к организации обслуживания систем. Раньше существовали «окна» ночного времени, в которых выполнялось резервное копирование, исправление ошибок, реорганизация БД, установка новых версий ПО, модернизация компьютера и ОС. Теперь это требуется делать в «горячем» режиме, совмещая с текущим обслуживанием пользователей. Решение этих задач требует новых подходов к организации вычислительных работ и управлению БД, при котором допустимо параллельное и бесконфликтное решение задач пользователей и администратора БД.

В современных условиях появляется потребность в обеспечении информационного обслуживания *мобильного* пользователя. Ранее типовой схемой ввода и обновления данных в базе являлся пакетный ввод информации и оперативный ввод с терминалов пользователей. Теперь нужно иметь возможность ведения БД как на центральной (стационарной) ВС, так и на портативном (переносном) компьютере. При этом необходимо иметь средства загрузки/выгрузки выбранных данных с центральной на портативную ЭВМ, а также средства обеспечения согласованности информации в обеих базах.

Определенные шаги в реализации такого рода услуг в некоторых СУБД уже сделаны. Так, в системе Access имеется средство реPLICATIONи БД (подраздел 10.9). Другим примером реализации средства работы с БД мобильного пользователя является СУБД Oracle Lite (Oracle), функционирующая в карманных компьютерах с операционной системой Windows CE. Эта СУБД, называемая клиентской, обеспечивает доступ через сеть к данным и приложениям, поддерживаемым системой Oracle 8. Категорию пользователей, для которых разработана Oracle Lite, составляют менеджеры, руководители и спе-

циалисты, находящиеся за пределами своих рабочих мест и требующие подключения к системным ресурсам. Им предоставляется возможность осуществлять синхронизацию локальных БД с информацией на сервере, выполнять запросы к корпоративным БД, запускать приложения для обработки данных на своем компьютере и на сервере.

9.3. Стандартизация баз данных

Необходимость стандартизации в области обработки данных осознана еще на этапе внедрения первых «массовых» СУБД. В настоящее время в индустрии информационных технологий работает более 250 подкомитетов официальных стандартизирующих организаций. Ими принято или находится в процессе разработки более 1000 стандартов. Рассмотрим характеристику стандартизации и наиболее распространенные стандарты для баз данных.

Характеристика процесса стандартизации

В процессе стандартизации сначала идет накопление опыта организаций разработчиков и предложений теоретиков. Затем выполняются обобщение, прогнозирование дальнейшего развития и выработка предложений по единым стандартам. После этого производится трудоемкое согласование документов представителями заинтересованных организаций. Потом осуществляется реализация принятых стандартов в программных разработках.

На принятие стандартов оказывают влияние многие факторы. Во-первых, разработкой стандартов занимаются несколько организаций и каждая из них в итоге принимает свой стандарт, чем-то отличающийся от других. Во-вторых, процесс разработки стандартов организациями идет параллельно с разработками, поэтому подготавливаемые документы постоянно модифицируются. В-третьих, не все ясно к моменту обсуждения и часто требуются научные исследования, затягивающие принятие решений. В-четвертых, на характер принимаемых решений оказывают влияние мощные организации компьютерной индустрии (элемент политики) и т. д. Все это, естественно, затягивает принятие стандартов на длительное время. Стандарты обычно имеют несколько версий с различными уточнениями, пояснениями, дополнениями и дополнительными документами.

Процесс претворения стандартов в жизнь для крупных корпораций оказывается более простым, т. к. они обычно участвуют в принятии стандартов. Более мелкие и молодые организации сталкиваются с проблемами интерпретации стандартизованных рекомендаций. Стандарты не могут быть одинаково хороши для всех практических задач, несмотря на усилия идеологов стандартов. Все это, безусловно, приводит к многообразию решений.

Короче говоря, наличие международных стандартов и рекомендаций не устраняет всех возможных противоречий и многообразия принимаемых фирмами-разработчиками решений. Польза от стандартов неоспорима потому, что без них было бы хуже, а часто и просто невозможно, решить вопросы международного уровня. Например, без принятия стандартов на связь, нельзя было бы организовать сеть Интернет.

Обычному пользователю и начинающему разработчику требуется иметь общее представление об имеющихся стандартах и их реализациях наиболее крупными фирмами. Это может помочь ему решить собственные задачи, не изобретая велосипед.

Реализации стандартов разными фирмами отличаются друг от друга, и все их особенности не описать. Познакомившись с наиболее важными стандартами, пользователь может легко использовать любую реализацию стандарта. В частности, это относится к стандартному языку доступа к БД – языку SQL. Средства его реализации в различных СУБД отличаются друг от друга незначительно.

Стандартизация в области баз данных затрагивает синтаксис и семантику языков баз данных, управление данными и транзакциями, организацию взаимодействия СУБД с другими элементами распределенных систем, параллельную обработку запросов, защиту данных от несанкционированного доступа и другие аспекты.

Рассмотрим с точки зрения стандартизации основной язык доступа к реляционным базам данных – язык SQL.

Стандартизация языка SQL

Язык SQL разработан в середине 70-х годов научно-исследовательской лабораторией фирмы IBM в рамках проекта реляционной СУБД System R. Исходное название языка – SEQUEL (Structured English QUEry Language). Язык основан на реляционном исчислении с переменными кортежами (подраздел 3.7). Кроме основных операторов формулирования запросов и манипулирования БД, он включал также средства описания и изменения схемы БД, определения ограничений целостности, защиты доступа (авторизации) к отношениям и их полям, получения точек сохранения транзакций, выполнения откатов и прочее.

Постепенно SQL был взят на вооружение различными организациями и фирмами-разработчиками ПО и появились стандарты SQL следующих организаций: ANSI (American National Standards Institute – Американский Национальный Институт Стандартов), SAG (SQL Access Group), X/Open (группа стандартов для UNIX), ISO (International Standard Organization – Международная организация по стандартизации), федерального правительства США, а также фирмы IBM.

Наиболее широко используемыми стандартами являются стандарт 1989 года ANSI SQL-89 и стандарты 1992 года: ANSI SQL-92 и ISO SQL-92.

Стандарт ANSI SQL-89 описывает три варианта использования SQL (три интерфейса): «модульный язык», «встроенный SQL» и «непосредственный вызов».

Модульный язык предусматривает возможность создания процедур, вызываемых из программ на традиционных языках программирования (С, Кобол, Фортран и другие).

Встроенный SQL предусматривает включение в программы на обычных языках программирования SQL-операторов. Совокупность SQL-операторов называют HOST-языком. Обработка программы с использованием SQL-операторов происходит в два этапа: сначала работает препроцессор, преобразующий SQL-операторы в некоторые команды языка программирования с учетом используемой СУБД, а затем – обычный компилятор с основного языка программирования. При включении SQL-операторов используется статический метод, означающий полное определение параметров операторов до выполнения программы.

Интерфейс на уровне непосредственных вызовов отличается от встроенного SQL тем, что исключена предварительная обработка препроцессором и используются внешние библиотеки функций.

Основной набор операторов SQL включает операторы определения схемы БД, выборки и манипулирования данными, управления транзакциями, авторизации доступа к данным, поддержки встраивания SQL в системы программирования и другие вспомогательные средства.

Несмотря на широкий набор возможностей, этот стандарт обладал функциональной неполнотой. Работа над SQL была продолжена, что привело к появлению в 1992 г. новых стандартов. Получили широкое применение и оказались по сути эквивалентными стандарты ISO SQL-92 и ANSI SQL-92.

Стандарт SQL-92 поддерживается практически всеми современными программами работы с базами данных. Он расширяет функции предыдущего стандарта, а главное, – предусматривает динамический SQL. Отличием динамического метода выполнения SQL-операторов от статического является то, что параметры SQL-операторов определяются непосредственно при выполнении программы. Несмотря на некоторое снижение быстродействия выполнения программы, динамический метод выполнения SQL-операторов позволяет повысить гибкость работы с базами данных и повысить независимость приложений от типов баз данных.

С принятием этого стандарта начата работа над новым стандартом SQL3.

Стандарт языка SQL3 разрабатывался ANSI совместно с ISO. Более точно: комитетом X3H2 по стандартам в области синтаксиса и семантики языков баз данных института ANSI (ANSI X3H2) и рабочей группой WG3 по базам данных подкомитета SC21, занимающегося взаимосвязью открытых систем, управлением данными и открытой распределенной обработкой. Объединенного комитета Международной организации по стандартизации и Меж-

дународной электротехнической комиссии (ISO/IEC JTC1). Кроме того, на разработку стандарта оказывали влияние организации, работающие в области объектно-ориентированных технологий, в частности: консорциум поставщиков программного обеспечения объектно-ориентированных баз данных ODMG (The Object Database Management Group) и консорциум по программному обеспечению, разрабатывающий спецификации объектно-ориентированных интерфейсов OMG (Object Management Group).

Текущий проект SQL3 практически полностью включает спецификацию SQL2, а также содержит ряд новых возможностей, в том числе две весьма существенных: объектные функциональные возможности и средства обеспечения вычислительной полноты языка – Управление хранимыми процедурами PSM (Procedure Storage Management).

К новой объектной возможности SQL3 прежде всего относится обеспечение объектной ориентированности таблиц БД. По сравнению с обычными реляционными таблицами теперь строки таблиц могут содержать абстрактные типы данных (АТД) и ссылки на АТД из других строк, а также иметь однозначные идентификаторы (вместо поиска строки по значениям). Поддержка АТД предусматривает средства определения типов и методов доступа к ним, аналогичных объектно-ориентированным языкам программирования типа C++, Smalltalk, Ada и других.

Напомним, что хранимые процедуры (подраздел 5.4) – это группы выполняемых операторов, которые хранятся не в приложении, а в базе данных. К разрешенным внутри хранимых процедур операторам относятся: операторы SQL, арифметические операторы, операторы передачи управления, описания функций и т. д.

Стандарт ODMG-93

Главной целью стандарта ODMG-93 является обеспечение независимости (мобильности) прикладных систем от систем управления объектными базами данных (СУБД). При разработке стандарта ставилась задача увязки решений с рекомендациями группы, занимающейся стандартизацией в области объектно-ориентированных языков программирования – OMG (Object Management Group).

Согласно ODMG-93, СУБД, с одной стороны, рассматривается как СУБД, в которой данные имеют объектную модель, а с другой – как средство представления объектов БД в качестве объектов ряда языков программирования. СУБД расширяют языки программирования, например, C++, средствами долговременного хранения объектов, управления конкурентным доступом, восстановления данных, выполнения объектных запросов и т. д.

В стандарте ODMG-93 дается определение следующих компонентов баз данных:

- объектной модели, которую должны поддерживать СУБД;

- языка определения объектов ODL (Object Definition Language);
- языка объектных запросов OQL (Object Query Language);
- языка манипулирования объектами OML (Object Manipulation Language), используемого в объектно-ориентированных языках программирования.

Схема взаимосвязи основных элементов стандарта ODMG показаны на рис. 9.3.

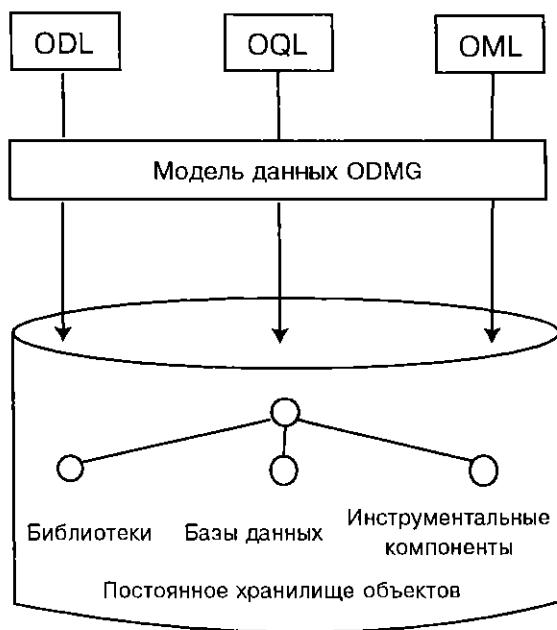


Рис. 9.3. Схема взаимосвязи основных элементов стандарта ODMG

За основу языка ODL в стандарте ODMG-93 взят язык определения интерфейсов IDL (Interface Definition Language) архитектуры OMG. В IDL каждый объект базы данных имеет строго определенный и единственный тип. Поведение объекта определяется перечнем допустимых операций в соответствии с типом. Объектная модель позволяет описывать связи типа 1:1, 1:M и M:M.

Язык объектных запросов OQL представляет собой декларативный язык работы с объектными БД. Синтаксической основой языка является SQL2. Основное отличие OQL от языка SQL состоит в том, что запрос формулируется в терминах объектов и коллекций (а не таблиц), а результатом являются объекты и коллекции (а не кортежи).

Операции манипулирования данными, управления транзакциями, динамического вызова запросов на OQL и другие действия выполняются с помощью библиотек классов и функций.

Постоянное хранилище объектов представляет собой интегрированную систему для согласованного хранения данных и программного кода. Логическая организация хранилища должна основываться на модели данных ODMG. Его физическая организация у различных СУБД может различаться, но она должна обеспечивать эффективные структуры данных для хранения иерархии типов и объектов, являющихся экземплярами этих типов.

Перечень инструментальных средств и библиотек в стандарте не определяется. Важным является объектно-ориентированный принцип их организации. Так, инструментальные средства поддержки разработки пользовательских приложений и их графических интерфейсов программируются на одном из OML и сохраняются как часть иерархии классов. Библиотеки функций (доступа, вычислений и т. д.) хранятся в иерархии типов и являются доступными из программного кода приложения.

Стандарт ODMG-93 используется при разработке СУБД и в качестве канонической объектной модели при проектировании систем управления неоднородными мультибазами данных. Примером использования стандарта является проект IRO-DB программы ESPRIT III.

Технология CORBA

Для информационных систем с распределенной обработкой в качестве стандарта группой OMG предложена технология CORBA (Common Object Request Broker Architecture – архитектура брокера общих объектных запросов). Спецификации написаны на пассивном языке описания интерфейсов IDL (Interface Definition Language), определяющем функциональность компонентов – внешние интерфейсы с потенциальными клиентами. Написанные на этом языке компоненты должны быть доступны независимо от языков, инструментальных средств, операционных систем и сетевой инфраструктуры программных компонентов.

Основу архитектуры CORBA составляет *брокер объектных запросов ORB* (Object Request Broker), управляющий взаимодействием клиентов и серверов в распределенной сетевой среде. Брокер объектных запросов представляет собой объектную шину, позволяющую объектам прозрачно генерировать запросы и получать отклики от других локальных и удаленных объектов.

Важнейшими свойствами CORBA ORB, определяющими ее достоинства для архитектуры клиент-сервер, являются следующие:

- возможность *статически* определять вызовы методов при компиляции или находить их *динамически* в процессе выполнения;
- возможность вызова методов серверного объекта с использованием *произвольного языка высокого уровня*. Поскольку CORBA отделяет интер-

фейс от его реализации и предоставляет независимые от языка типы данных, это дает возможность вызывать объекты из любого языка и для любой ОС;

- **самоописываемость** системы заключается в предоставлении метаданных на этапе выполнения для описания каждого известного серверного интерфейса;
- **прозрачность** локальная и удаленная состоит в том, что ORB может выполняться в автономном режиме или взаимодействовать с другими ORB с помощью сервисов протокола ИОР (Internet Inter-ORB Protocol – Интернет-протокол взаимодействия ORB);
- **встроенная безопасность и механизм транзакций** основаны на включении в сообщения соответствующей контекстной информации;
- **возможность полиморфных сообщений** основана на реализации вызова функции целевого объекта;
- **возможность инкапсуляции** существующих приложений, основанная на отделении определения объекта от его реализации в архитектуре системы.

В общем, CORBA можно определить как промежуточное программное обеспечение, потенциально связанное с другими формами клиент-серверных видов программного обеспечения. Программное обеспечение CORBA разработано для всех основных аппаратных и программных платформ. Поэтому клиентские приложения CORBA одной операционной системы легко взаимодействуют с сервером приложений в любой другой операционной системе.

Версия технологии CORBA Real-Time ориентирована на поддержку функционирования и повышение производительности распределенных систем в масштабе реального времени. С ее помощью предполагается задавать продолжительность функционирования отдельных компонентов и приоритеты выполнения, планировать работу компонентов и т. д.

Стандарты в продуктах ведущих фирм

Рассмотрим, как действующие стандарты используются в программных продуктах ведущих фирм IBM, Microsoft и Borland.

Фирма IBM для обеспечения мобильности (переносимости) операционных систем разработала архитектуру SAA (System Application Architecture).

Архитектура SAA предполагает наличие **унифицированных** средств разработки приложений, доступа к БД и телекоммуникационного доступа. Кроме того, у пользователей должна быть возможность использовать готовые приложения других организаций. Согласно архитектуре SAA, системные программные средства предоставляют три интерфейса:

- доступа конечного пользователя;
- прикладного программирования;
- телекоммуникационный (связи с удаленными пользователями и объектами сети).

Интерфейс доступа конечного пользователя охватывает комплекс технических, организационных и программных решений, обеспечивающих простоту и унификацию взаимодействия конечного пользователя с различными моделями ЭВМ.

Интерфейс прикладного программирования, кроме стандартных языков и средств программирования, содержит генератор приложений, языки командных процедур, языки баз данных и запросов (SQL и QMF) и средства обеспечения диалога.

В качестве телекоммуникационного интерфейса предлагаются протоколы сетевой архитектуры SNA (System Network Architecture – сетевая архитектура системы), средства выхода в сети, поддерживающие протокол X.25, а также протоколы потоков данных для устройств типа IBM 3270.

Фирма Microsoft на основе стандарта CAE разработала открытый интерфейс ODBC (Open DataBase Connectivity – совместимость открытых баз данных) доступа к базам данных из приложений.

Стандарт CAE (Common Application Environment – общая прикладная среда) был разработан группой SAG и X/Open с целью разрешить противоречия множества схожих существующих стандартов прикладного программирования. Этот стандарт опирается на стандарты ANSI, ISO и собственные стандарты групп. Основное его назначение – обеспечение взаимодействия прикладных программ в открытых системах.

ODBC представляет собой интерфейс прикладного программирования в виде библиотеки функций, вызываемых из различных программных сред и позволяющих приложениям унифицированно обращаться на SQL к базам данных различных форматов.

Функции интерфейса ODBC включают следующие шесть основных групп:

- назначение идентификаторов окружения, соединения и SQL-операторов;
- соединение;
- выполнение SQL-операторов;
- получение результатов;
- управление транзакциями;
- идентификация ошибок и смешанные функции.

При выполнении приложений, использующих ODBC, предусматриваются следующие этапы:

- инициализация (назначение идентификаторов окружения и соединений, соединение с сервером, назначение идентификаторов SQL-операторам);
- выполнение SQL-операторов с анализом результатов;
- завершение (освобождение идентификаторов SQL-операторов, разрыв соединения, освобождение идентификаторов соединений и окружения).

Интерфейс ODBC поддерживает общий набор функций языка SQL для доступа к базам данных и позволяет работать в так называемом *прозрачном*

режиме по соглашениям конкретной базы данных. Кроме того, он содержит уровни функциональности или иерархии функциональных возможностей: минимальную, основную и расширенную.

Программные средства поддержки ODBC корпорация Microsoft обычно поставляет вместе с СУБД. Так, вместе с MS Access предоставляется менеджер драйверов и драйвер доступа к MS SQL Server. Драйверы доступа к форматам других БД берутся от фирм-разработчиков соответствующих СУБД.

Фирма Borland в основных программных продуктах Delphi, Paradox for Windows, dBase for Windows и других, использует собственное стандартизованное средство доступа к базам данных *Borland Database Engine (BDE)* – процессор баз данных фирмы Borland.

BDE включает следующие три основных компонента: стандартный интерфейс доступа к базам данных *IDAPI* (Integrated Database Application Program Interface), драйверы баз данных распространенных форматов и утилиты настройки драйверов и псевдонимов.

Интерфейс IDAPI насчитывает более 150 функций доступа к различным БД и позволяет просто и единообразно работать с локальными и с удаленными данными. В основе механизма доступа лежит понятие курсора (подраздел 3.9). IDAPI не ограничен минимальным набором функций, поддерживаемых БД. Как и в ODBC, здесь поддерживается расширенное множество функций в соответствии с возможностями форматов локальных и удаленных БД.

Важной особенностью интерфейса IDAPI является обеспечение программ пользователя средствами вторичного уточнения результатов запросов – фильтрами. Функционируя в среде клиентской части приложения, фильтры позволяют при работе с удаленными данными уменьшить объем передаваемой по сети информации. Фильтры IDAPI позволяют описывать сложные логические условия над данными БД.

Интерфейс IDAPI обеспечивает преобразование своих вызовов в вызовы функций интерфейса ODBC. Для обращения к функциям IDAPI можно пользоваться языками запросов SQL (из программ) и QBE (в диалоговом режиме).

9.4. Характеристика технологии ADO.NET

В этом подразделе рассматривается технология доступа к данным ADO.NET, указаны ее преимущества по сравнению с предыдущей технологией ADO, а также дана характеристика ее объектной модели. Технология ADO.NET является развитием технологии доступа к данным ADO и входит составной частью в Microsoft .NET.

Характеристика Microsoft .NET

Соответственно, возникает вопрос, что представляет собой сама Microsoft .NET. Прежде всего, это новый подход к разработке приложений, при котором разработчику, как правило, не нужно самостоятельно писать код, определяющий его функциональность. Вместо этого разработчику достаточно воспользоваться соответствующей службой (сервисом) сети Интернет. В состав Microsoft .NET входят следующие основные компоненты: библиотека классов .NET; языки .NET; общая языковая среда исполнения (CLR); интегрированная среда разработки приложений, например, Visial Studio .NET. Дадим краткую характеристику названным компонентам:

- Библиотека классов .NET включает множество шаблонов для использования их в различных приложениях. Любой язык .NET может использовать возможности этой библиотеки, взаимодействуя с соответствующим набором объектов.
- Common Language Runtime (CLR) – общеязыковая исполняющая среда, обеспечивающая единообразный доступ к сервисам Microsoft .NET. Основной задачей CLR является загрузка и выполнение кода, написанного на любом языке программирования, поддерживаемом в Microsoft .NET. В настоящее время поддерживаются языки Visual Basic, C++, C#, JScript из состава продуктов фирмы Microsoft, а также языки других фирм, в числе которых COBOL, Eiffel, Oberon, Perl, SmallTalk и др.

Технология ADO.NET

ADO.NET (ActiveX Data Object .NET) – набор классов, используемый для доступа к источникам данных в платформе .NET. ADO.NET представляет собой новую объектную модель, которая использует стандарт XML для передачи данных. В ADO.NET реализована идея использования *отсоединенных* наборов данных, причем такой способ работы является основным. По сравнению с ADO, ADO.NET предполагает более легкое программирование, лучшую производительность и масштабирование, меньшую зависимость от источников данных.

ADO.NET можно использовать для доступа к реляционным СУБД, таким как SQL Server 2000, и ко многим дополнительным источникам данных, для работы с которыми предназначен провайдер OLE DB.

Одно из ключевых новшеств ADO.NET – замена ADO-объекта RecordSet комбинацией объектов DataTable, DataSet, DataAdapter и DataReader. Объект DataTable представляет набор (collection) записей отдельной таблицы и в этом отношении аналогичен RecordSet. Объект DataSet представляет набор объектов DataTable, а также содержит отношения и ограничения, используемые при связывании таблиц. На самом деле DataSet – это хранящаяся в памяти реляционная структура данных со встроенной поддержкой XML.

В ADO.NET поддержка реляционной модели осуществляется двумя способами. Первый способ реализует подсоединеную модель доступа к данным,

в которой функционируют стандартные методы доступа к реляционной базе данных, включая поддержку параметрических запросов, хранимых процедур, SQL-операторов пакетного выполнения и транзакций. Этот способ полностью аналогичен способам, применяющимся в известных интерфейсах доступа к данным, таких, как OLE DB, ODBC, JDBC.

При *втором* способе доступ к данным осуществляется в *отсоединенном* режиме с помощью объекта **DataSet**, имитирующего базу данных. Над данными, содержащимися в этом объекте применимы все операции, характерные для баз данных. Объект **DataSet** может содержать любое количество реляционных таблиц, отношений и ограничений и даже позволяет получать подмножество таблиц с помощью SQL-подобного языка запросов. Кроме того, в отсоединенную модель входит объект **DataAdapter**, который играет роль связующего класса между реальной базой данных и объектом **DataSet**.

Преимущества ADO.NET

Технология ADO.NET создана для использования в управляемых проектах. Предыдущая технология ADO основана на технологии СОМ и при использовании из управляемых приложений требует дополнительных затрат на выполнение кода. К тому же ADO имеет меньшие возможности при работе с отключенными наборами данных и XML. Например, в ADO непросто сохранить изменения, произведенные в отключенном курсоре. Рассмотрим некоторые преимущества ADO.NET в сравнении с ADO.

Масштабируемость. При использовании объекта **DataSet** работа происходит с отсоединенными наборами данных. Это означает, что вы используете соединение с источником данных очень короткое время. Во многих системах количество подключений к базам данных является самым узким местом в плане масштабируемости. Для таких систем ADO.NET является хорошим решением, резко повышающим их масштабируемость.

Независимость от источника данных. В ADO возможности объекта **RecordSet** сильно зависели от используемого источника данных. Теоретически ADO обеспечивал доступ к данным независимо от источника данных, на практике всегда необходимо было иметь хорошее представление о возможностях провайдера. В ADO.NET **DataSet** действительно не зависит от источника данных, и изменение провайдера, с помощью которого заполняется **DataSet**, не влияет на функциональность **DataSet**.

Способность к взаимодействию. Так как ADO.NET использует XML как стандартный формат передачи данных, программа, которой необходимо получить данные из компонента ADO.NET, не обязана сама быть компонентом ADO.NET. В общем случае она вообще может не быть Windows-программой. Единственное требование – эта программа должна понимать XML. И это позволяет ADO.NET-компонентам при использовании других компонентов и служб легко взаимодействовать с любой программой на любой платформе.

Компоненты ADO.NET

Компоненты ADO.NET существуют в пяти главных пространствах имен в библиотеке классов .NET. Укажем назначение этих пространств имен.

System.Data – содержит фундаментальные классы с базовой функциональностью ADO.NET. В их число входят классы **DataSet** и **DataRelation**, которые позволяют манипулировать структуризованными реляционными данными.

System.Data.Common – эти классы применяются другими классами в пространствах имен **System.Data.SqlClient** и **System.Data.OleDb**, которые являются их наследниками и предоставляют конкретные версии, настроенные для поставщиков SQL Server и OLE DB.

System.Data.OleDb – содержит классы, которые используются для соединения с поставщиком OLE DB, включая **OleDbCommand** и **OleDbConnection**.

System.Data.SqlClient – содержит классы, которые используются для соединения с базой данных Microsoft SQL Server. Эти классы предоставляют такие же свойства и методы аналогично классам из **System.Data.OleDb**. Отличие состоит в том, что они оптимизированы для SQL Server.

System.Data.SqlTypes – содержит структуры для специфических типов данных SQL Server, например, **SqlMoney** и **SqlDateTime**. Эти типы не являются обязательными и используются для работы с типами данных SQL Server без необходимости конвертировать их в стандартные эквиваленты .NET. Использование таких структур позволяет увеличить производительность обмена данными за счет устранения автоматического преобразования данных.

Объекты ADO.NET

Базовые объекты ADO.NET можно разбить на две группы: первая используется для хранения и управлениями данными (классы **DataSet**, **DataTable**, **DataRow** и **DataRelation**), вторая группа – для обеспечения соединения с источниками данных (классы **Connections**, **Commands** и **DataReader**). Объекты ADO.NET второй группы существуют в двух видах: для связи с поставщиками OLE DB и для взаимодействия с SQL Server.

Объекты данных позволяют хранить локальные данные, не имеющие связи с источником. Кроме того, они не хранят соединение с источником данных и, следовательно, их можно создавать самостоятельно, не используя базу данных. Объект **DataSet** не имеет непосредственного соединения с источником данных и его можно создавать без обращения к базе данных.

Схема доступа к данным

Одна из схем доступа к данным с помощью ADO.NET представлена на рис. 9.4. Необходимая пользователю информация находится в базе данных. Обрабатываемые данные размещаются в объекте **DataSet**, не имеющем соединения с источником данных. Тем самым достигается возможность работы с данными в течение длительного промежутка времени. На рис. 9.4 се-

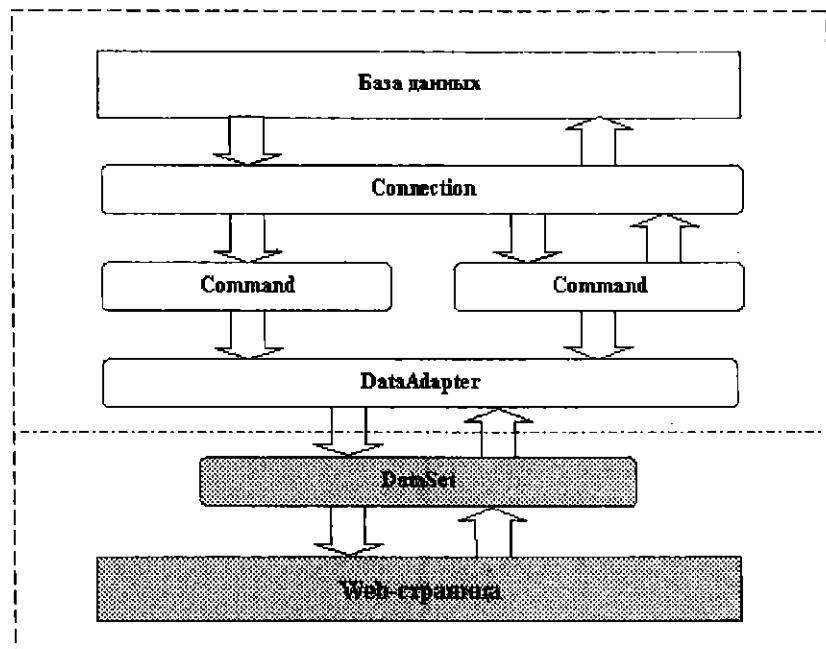


Рис. 9.4. Схема доступа к данным с помощью ADO.NET

рым цветом под пунктирной линией для наглядности выделены не подсоединеные объекты.

В заключение отметим, что технология ADO.NET представляет собой концепцию универсального доступа к данным, которую можно определить как распределенное хранение данных и управление ими.

Контрольные вопросы и задания

1. Охарактеризуйте подход к выбору СУБД.
2. Назовите основные показатели пригодности программных продуктов.
3. Перечислите показатели пригодности СУБД.
4. Приведите примеры основных технических характеристик СУБД.
5. Назовите основные виды тестов, используемых для оценки производительности.
6. Назовите состав вопросов, связанных с определением структуры программно-аппаратного обеспечения.
7. Укажите факторы, которые нужно учитывать при выборе сервера информационной системы.
8. Как определяется размер основной памяти компьютера для серверной части информационной системы?

9. Укажите эмпирические правила для определения размера кэша.
10. Что учитывают при подборе дисковой системы ввода/вывода?
11. Укажите разновидности сильносвязанных вычислительных систем.
12. Охарактеризуйте слабосвязанные вычислительные системы.
13. Назовите методы распараллеливания обработки данных.
14. Перечислите возможные направления развития СУБД.
15. Дайте общую характеристику процесса стандартизации.
16. Перечислите стандарты языка SQL и дайте краткую их характеристику.
17. Каково назначение стандарта ODMG-93?
18. Охарактеризуйте технологию CORBA.
19. Назовите стандарты, используемые в продуктах фирм IBM, Microsoft и Borland.
20. Укажите основные компоненты Microsoft .NET.
21. В чем заключаются достоинства ADO.NET?
22. Охарактеризуйте схему доступа к данным с помощью ADO.NET.

Литература

1. *Волков А. А. Тесты ТРС // Системы Управления Базами Данных, № 2, 1995. С. 70–78.*
2. *Базы данных: достижения и перспективы на пороге 21-го столетия / Под ред. Ави Зильбершатца, Майкла Стоунбрайкера и Джейфа Ульмана // Системы Управления Базами Данных, № 3, 1996. С. 103–117.*
3. *Дейт К. Дж. Введение в системы баз данных / Пер. с англ. – 6-е изд. К.: Диалектика, 1998.*
4. *Дэвидт Т., Грей Д. Параллельные системы баз данных: будущее высокоэффективных систем баз данных // Системы Управления Базами Данных, № 2, 1995. С. 8–31.*
5. *Калиниченко Б. О. Асинхронное тиражирование данных в гетерогенных средах // Системы Управления Базами Данных, № 3, 1996. С. 118–124.*
6. *Кузнецов С.Д. Три манифеста баз данных: ретроспектива и перспективы. Институт системного программирования РАН - Информационно-аналитические материалы Центра Информационных Технологий. - <http://www.citforum.ru>.*
7. *Меллинг В. П. Корпоративные информационные архитектуры: и все-таки они меняются // Системы Управления Базами Данных, № 2, 1995. С. 45–59.*
8. *Михайлов М. СУБД нового поколения // КомпьютерПресс, № 11, 1990. С. 25–30, 79.*
9. *Орфали Р., Харки Д., Эдвардс Д. Основы CORBA / Пер. с англ. М.: МАИП, Горячая линия – Телеком, 1999.*
10. *Просиз Д. Программирование для Microsoft .NET /Пер. с англ. – М.: Издательско-торговый дом «Русская редакция», 2003. – 704 с.*
11. *Сигнор Роберт, Стегман Михаэль О. Использование ODBC для доступа к базам данных / Пер. с англ. М.: БИНОМ; НАУЧНАЯ КНИГА.*

12. Системы управления базами данных и знаний: Справ. изд. / Наумов А. Н., Вендрев А. М., Иванов В. К. и др.; Под ред. А. Н. Наумова. М.: Финансы и статистика, 1991.
13. Хомоненко А. Д., Гофман В. Э. Работа с базами данных в Delphi. – 3-е изд. – СПб.: БХВ-Петербург, 2005.
14. Шнитман В. З., Кузнецов С. Д. Серверы корпоративных баз данных. Информационно-аналитические материалы Центра Информационных Технологий. <http://www.citforum.ru>.

СОВРЕМЕННЫЕ СУБД И ИХ ПРИМЕНЕНИЕ

10. СУБД Access 2002

10.1. Общая характеристика

Программа Microsoft Access 2002 является реляционной СУБД, которая может функционировать под управлением операционных систем Windows 95/98, Windows NT 3.51 и выше, а также более новых версий Windows (в том числе Windows 2000 и Windows XP). Она является дальнейшим развитием предыдущей версии Microsoft Access 2000.

СУБД Access имеет стандартизованный интерфейс приложений Windows (рис. 10.1). Большинство действий по работе с различными элементами в среде Access можно выполнить с помощью следующих средств: команд основного меню, кнопок панели инструментов, команд контекстного меню и комбинаций клавиш. Для краткости будем обычно указывать первый или первые два

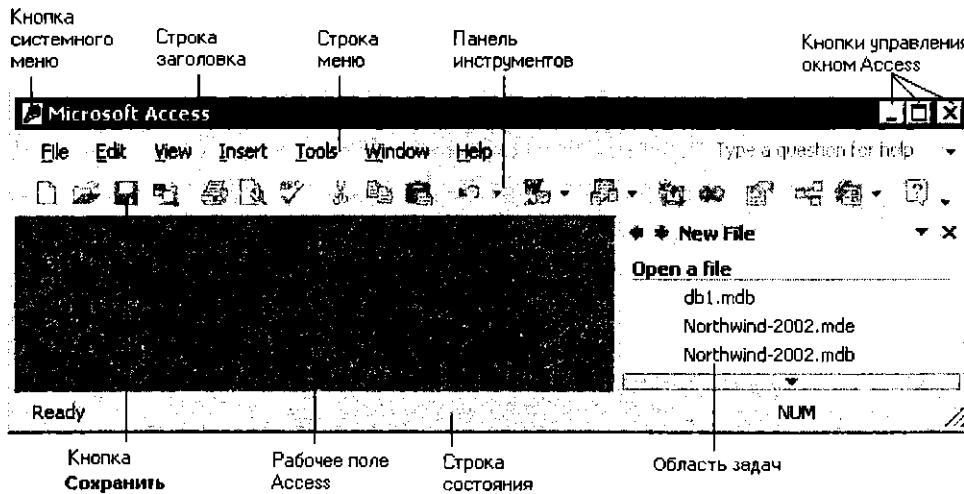


Рис. 10.1. Окно Microsoft Access

способа. Отметим, что контекстное меню определенного элемента БД вызывается щелчком правой кнопкой мыши при размещении ее указателя над этим элементом.

База данных является основным компонентом проекта приложения Access и может включать в свой состав таблицы, формы, запросы, отчеты, макросы и модули.

Для работы с базами данных в Access имеется стандартное окно, из которого можно вызвать любой ее объект просмотра, выполнения, разработки или модификации. Пользователь для работы с базой данных может разработать свой интерфейс, основу которого обычно составляют *формы*. На формах размещаются различные элементы, такие как: поля таблиц, поля со списком, кнопки, раскрывающиеся списки, выключатели, переключатели, флаги, рисунки, подчиненные формы и т. д.

За кнопками обычно закрепляют вызов функций. Функции обработки информации во время работы с базой данных задаются с помощью макросов или программ на Visual Basic for Application (VBA) – VBA-программ. Обычно в приложениях для работы с БД предусматривают автоматическое открытие главной кнопочной формы при открытии базы данных. В последующем работа пользователя происходит с помощью главной формы и при необходимости с помощью других форм и окон. Базу данных, имеющую интерфейс с пользователем, можно считать приложением, поскольку все описания базы данных (в том числе программные коды) интерпретируются системой Access при работе пользователя.

Текущая открытая БД может взаимодействовать с внешними БД, которые используются как источник таблиц при импорте или присоединении, а также как получатель при экспорте данных из текущей базы данных. С помощью запросов во внешней БД можно создавать таблицы. В качестве внешней БД может выступать любая база данных, поддерживающая протокол ODBC, например, база данных SQL Server, расположенная на удаленном сервере, или одна из баз данных систем Paradox, dBASE, или Access.

Таблица представляет собой основную единицу хранения данных в базе. Понятие таблицы в Access полностью соответствует аналогичному понятию реляционной модели данных. В произвольной базе обычно имеется совокупность связанных между собой таблиц. Между двумя таблицами можно устанавливать связи типа 1:1 и 1:M с помощью окна описания схемы данных. Основными операциями над таблицами являются: просмотр и обновление (ввод, модификация и удаление), сортировка, фильтрация и печать.

Форма представляет собой объект базы данных Access, в котором разработчик размещает элементы управления, принимающие действия пользователей или служащие для ввода, отображения и изменения данных в полях.

Запрос представляет собой формализованное требование на отбор данных из таблиц или на выполнение определенных действий с данными. Запрос позволяет создать набор записей из данных, находящихся в разных таблицах, и использовать его как источник данных для формы или отчета.

В Access можно создавать и выполнять следующие основные типы запросов: на выборку, обновление, удаление, или добавление данных. С помощью запросов можно также создавать новые таблицы, используя данные из одной или нескольких существующих таблиц.

Описание запроса можно выполнить с помощью бланка QBE или инструкции языка SQL.

Макрос представляет последовательность макрокоманд встроенного языка Access, задающих автоматическое выполнение некоторых операций, например: «Открыть Таблицу» (OpenTable), «Закрыть» (Close), «Найти Запись» (FindRecord) и «Печать» (PrintOut). В последующих версиях системы макросы используются для обеспечения совместимости с предыдущими версиями, и рекомендуется для процедур автоматизации использовать программный код VBA.

Модуль представляет совокупность описаний, инструкций и процедур на языке VBA, сохраненную под общим именем. В Access используются модули трех типов: формы, отчета и стандартный. Модули форм и отчетов содержат программы, являющиеся локальными для этих объектов. Процедуры из стандартного модуля, если они не описаны явно как локальные для содержащего их модуля, распознаются и могут вызываться процедурами из других модулей в той же базе данных или из адресуемых баз данных.

При работе с программой Microsoft Access пользователь может встретиться еще с двумя важными понятиями: *страница доступа к данным* и *проект*. Они не являются компонентами базы данных, хотя и имеют к ней отношение. Эти понятия будут рассмотрены отдельно позднее.

Access поддерживает традиционные для офисных приложений механизмы связывания и встраивания объектов OLE (Object Linking and Embedding) и динамического обмена данными DDE (Dynamic Data Exchange). При этом по протоколу OLE система Access позволяет работать с любыми объектами из библиотеки типов другого приложения, а также предоставляет свои объекты для других приложений.

Аналогично системой Access поддерживается протокол динамического обмена данными в роли приложения, принимающего данные (клиента), и источника данных (сервера). Например, база данных Access может быть сервером для приложения Microsoft Word, выступающего в роли клиента и принимающего данные по каналу связи.

Основные взаимосвязи объектов в БД Access схематично представлены на рис. 10.2 (взаимодействия объектов показаны сплошными линиями, потоки данных — штриховыми).

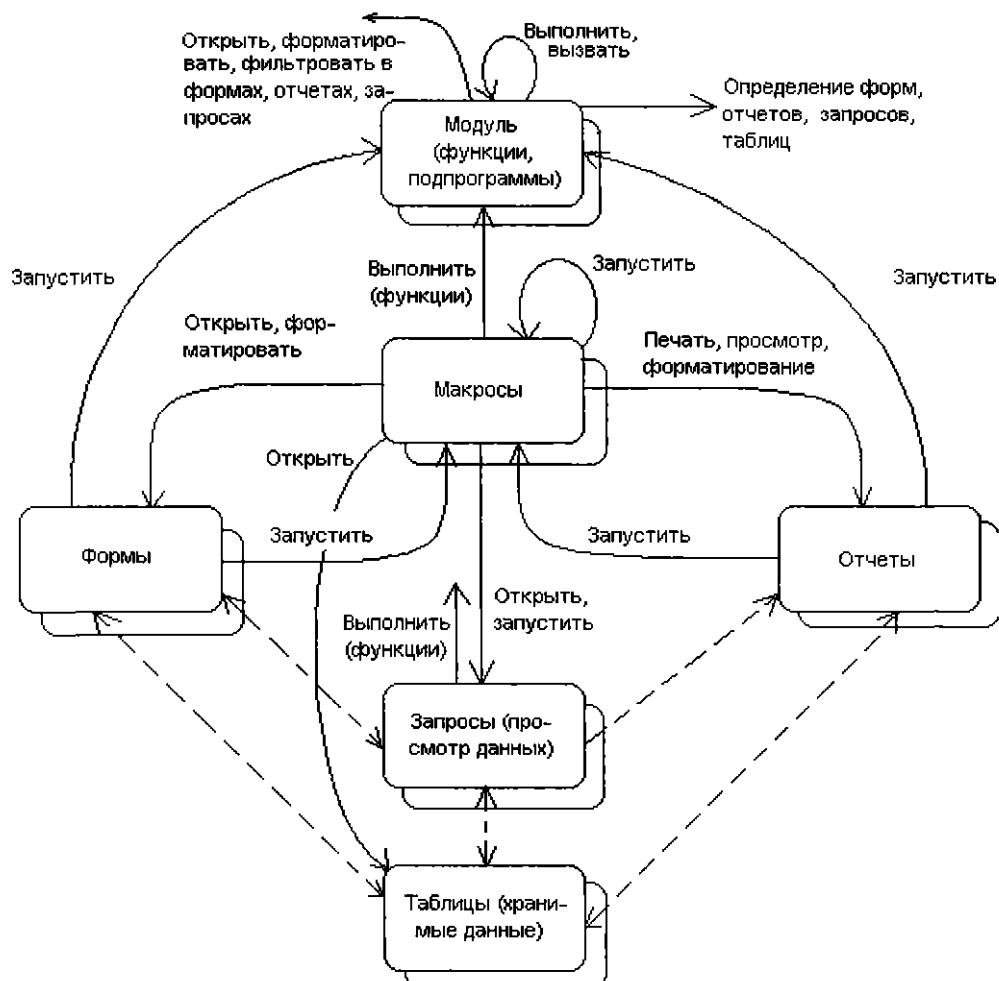


Рис. 10.2. Схема взаимосвязи объектов

В таблицах хранятся данные, которые можно использовать в запросах, формах и отчетах. Формы и отчеты используют данные из таблиц или через запросы (путем выборки). Операции вычисления или форматирования данных при выполнении запросов могут производиться с помощью встроенных функций или пользовательских VBA-программ. Макросы и VBA-программы могут вызываться при возникновении следующих событий в БД: открытие формы, ввод записи в таблицу, нажатие кнопки в окне формы и т. д. С помощью макросов можно, к примеру, открыть форму или отчет, выполнить запрос и прочее. VBA-программы могут работать с объектами Access и вызывать процедуры из DLL-библиотек Windows.

При полной установке Access версии 2002 требуется 16 Мб оперативной памяти и около 65 Мбайт свободного пространства на жестком диске.

Основной учебной базой данных является БД «Борей» (доступна при помощи команды **Примеры баз данных** в меню **Справка**). Она содержит данные о продажах вымышленной компании, занимающейся импортом/экспортом продуктов питания. Пользователь может не только просматривать объекты базы данных, но и выполнять различные операции с данными, поскольку она уже содержит достаточное количество записей со смысловой информацией.

В Microsoft Access 2002 имеется также учебный проект Access Northwind (NorthwindCS.adp), для которого базу данных можно создать при первом открытии проекта (команда **Примеры баз данных** в меню **Справка**) вручную либо автоматически с помощью Access.

Кроме того, в Microsoft Access включены три учебных приложения («Адресная книга», «Управление контактами» и «Личное имущество») с соответствующими базами данных, которые можно использовать для изучения продукта и личных нужд.

Изучение структуры и поведения во время работы с учебными базами данных и проектами позволяет лучше понять, как следует организовать и разрабатывать собственные базы и проекты, а также как использовать Microsoft Access.

Microsoft Access 2002 позволяет работать с данными и таблицами баз данных, созданных в предыдущих версиях Access, но его нельзя использовать для изменения объектов в базах данных предыдущих версий. Поэтому для работы с ранее созданными базами рекомендуется использовать имеющиеся в Microsoft Access 2002 средства преобразования старых форматов в новый формат.

Некоторые ограничения СУБД Access 2002:

- размер файла базы данных (с расширением mdb) – 2 Гб за вычетом места, необходимого системным объектам. Реально размер ограничивается доступным местом на диске, так как БД может включать присоединенные таблицы;
- число объектов в базе данных – 32768;
- количество одновременно работающих пользователей – 255;
- максимальный размер таблицы – 2 Гбайт;
- максимальное количество полей в таблице – 255;
- максимальное количество индексов в таблице – 32;
- максимальное число символов в записи (не считая поля Memo и поля объектов OLE) – 2000;
- максимальное число символов в поле Memo – 65 535 при вводе данных через интерфейс пользователя и 1 Гбайт при программном вводе данных;
- максимальный размер объекта OLE – 1 Гбайт;
- максимальное количество таблиц в запросе – 32.

Для Microsoft Access 2002 имеются дополнительные средства разработки и обслуживания, которые входят в состав пакета Microsoft Office XP для разработчиков. К пакету прилагается соответствующая документация — «Руководство разработчика Microsoft Office XP».

В число основных средств, имеющих непосредственное отношение к Access 2002, входят:

- средства, позволяющие распространять копии исполняемого приложения Microsoft Access всем пользователям, даже если на их компьютерах не установлен Microsoft Access;
- менеджер репликации (Replication Manager), позволяющий планировать обновления реплик, определять объекты, реплицированные в базе данных, и одновременно управлять несколькими наборами реплик;
- мастер Packaging Wizard, позволяющий распространять приложения баз данных и программируемые приложения Microsoft Office в сети или на компьютеры, где выполняется Visual Basic для приложений 6.0.

10.2. Новые возможности Microsoft Access 2002

Программа Microsoft Access 2002 (версия 10) продолжает линию Access предыдущих версий и имеет с ними много сходства. Подавляющее большинство функций, реализованных в Access 2000, сохранилось без серьезных изменений. Рассмотрим лишь наиболее важные из отличий.

Сводные таблицы и сводные диаграммы

В Microsoft Access 2002 появилась возможность открывать таблицы, запросы, представления, хранимые процедуры (в справочной системе русифицированного варианта Access называемые сохраненными процедурами), функции и формы в режимах сводной таблицы и сводной диаграммы. Это более удобно для анализа данных и создания сложных сводных таблиц и диаграмм. Существует возможность сохранять представления в режимах сводной таблицы и сводной диаграммы в качестве страниц доступа к данным, которые затем можно просмотреть с помощью браузера Microsoft Internet Explorer 5 или более поздней версии. Кроме того, подчиненные формы в режимах сводной таблицы и сводной диаграммы можно использовать точно так же, как они использовались ранее в режиме таблицы.

Поддержка языка XML

Язык XML (Extensible Markup Language) является в настоящее время не только стандартной технологией передачи данных в Интернете, но и предпочтительной технологией обмена данными между приложениями. Поэтому Microsoft Access 2002 предоставляет мощные, интуитивные способы совмес-

тного использования данных XML, независимо от платформы, формата данных, протокола, схемы и других особенностей. С помощью знакомого пользователю интерфейса он позволяет легко создавать данные и структуры документов XML, используя структуры и данные Jet или SQL Server. Кроме того, в формах, отчетах и на страницах доступа к данным можно использовать данные XML из других источников (внутренних серверах SQL, электронных таблицах и других средствах доступа, таких как SAP). Имеется также возможность создавать и применять схемы и таблицы стилей.

Поддержка расширенных свойств БД Microsoft SQL Server 2000

Интеграция Microsoft Access 2002 с Microsoft SQL Server позволила включить в проект Microsoft Access расширенные свойства базы данных SQL Server. Это, в свою очередь, сделало возможным использование таких средств как связи подстановок, условия на значения (вид ограничений целостности), форматирование текста и подгаблизы. Расширенные свойства можно использовать в таблицах, представлениях и хранимых процедурах аналогично другим объектам БД. С их помощью упростился перенос настроек столбцов, строки, шрифтов и масок данных из одного сеанса проекта Microsoft Access в другой. Кроме того, введение расширенных свойств упростило перемещение приложения из базы данных Microsoft Access в проект Microsoft Access, подключенный к Microsoft SQL Server.

Обновление страниц доступа к данным

Если проект Microsoft Access имеет страницы доступа к данным и подключен к локальному ядру Microsoft SQL Server 2000 Desktop Engine (ранее MSDE), то имеется следующая возможность. Можно сделать эти страницы доступными в автономном режиме, потом независимо вносить в них изменения на переносном компьютере и, наконец, выполнить автоматическую синхронизацию данных при подключении к серверу SQL.

Рассмотрим наиболее важные вопросы изучения системы Access, связанные с проектированием, разработкой и администрированием баз данных. Некоторые из них, например, защита информации в БД и создание автономно исполняемых приложений в литературе раскрыты недостаточно полно и вызывают определенные трудности.

10.3.Средства поддержки проектирования

Разработчикам приложений предыдущих поколений СУБД приходилось разрабатывать структуру БД, опираясь на теоретический материал. Средства разработки в СУБД были недостаточно совершенны. В большинстве случаев это были обычные языки программирования с внешними библиотеками подпрограмм.

В помощь пользователю при разработке структур БД в современных СУБД часто предлагаются учебные приложения БД с текстовыми описаниями и встроенным справочниками. В описаниях имеется информация о содержимом таблиц, связях их друг с другом, приводятся тексты программ и объектов разработки. С их помощью пользователь быстрее и эффективнее решает свою задачу, при необходимости используя готовые приемы.

Более интеллектуальные СУБД, в том числе Access, дополнительно предоставляют средства для предотвращения *аномалий*. К ним можно отнести *средство помощи при создании таблиц* (вспомогательное средство, не используемое отдельно) и *Мастер анализа таблиц* (основное средство).

Напомним, что перед созданием БД ее нужно спроектировать. Определить, сколько должно быть таблиц, как они между собой связаны, какие у каждой из таблиц поля, есть ли ключи и прочее. Бывают ситуации, когда: о разрабатываемой информационной системе собрана не вся информация; информация собрана, но не систематизирована; данные, которые необходимо распределить по таблицам БД, импортированы из других СУБД или из текстовых файлов и др. В подобных случаях требуется анализ информации. Если объем ее значителен, обойтись без средств автоматизации очень сложно.

Средство помощи создания таблиц БД при завершении создания таблицы предупреждает о желательности задать ключи в таблице, если они отсутствуют. Иметь ключи в таблице рекомендуется по двум причинам. Во-первых, при необходимости установления связей с другими таблицами. Во-вторых, определение ключевых полей гарантирует хранение информации в таблице как минимум во второй нормальной форме, что уменьшает риск появления аномалий.

Мастер анализа таблиц системы позволяет выполнять анализ и нормализацию таблиц. При этом исходная таблица, данные в одном или нескольких полях которой повторяются, разделяется на несколько связанных таблиц. Разделение таблиц происходит так, чтобы информация не терялась. Пользователь может создать таблицы самостоятельно или с помощью Мастера.

Для выполнения анализа таблицы и/или ее нормализации нужно открыть окно БД и выбрать в меню пункт Сервис | Анализ | Таблица (Tools | Analyze | Table). В результате запускается Мастер анализа таблиц. Возможность его использования предоставляется также после операций импорта данных из текстовых файлов или файлов электронных таблиц с помощью одного из Мастеров импорта.

При анализе таблиц с помощью Мастера с целью нормализации исходной таблицы пользователю разрешается следующее: переименовывать новые таблицы, устанавливать и добавлять ключи, отменять предыдущие операции и выполнять некоторые другие действия.

Проиллюстрируем работу Мастера анализа таблиц на примере таблицы «Устройства-Производители», содержимым которой является результат соединения таблиц О2 и Д2, связанных по типу 1:М (см. подраздел 3.3). Получим из этой таблицы ее исходные таблицы – О2 и Д2.

Таблица «Устройства-Производители»

Код	Вид устройства	Фирма-производитель	Наличие
а	CD-ROM	Acer	Да
а	CD-ROM	Mitsumi	Нет
а	CD-ROM	NEC	Да
а	CD-ROM	Panasonic	Да
а	CD-ROM	Sony	Да
б	CD-Recorder	Philips	Нет
б	CD-Recorder	Sony	Нет
б	CD-Recorder	Yamaha	Да
в	Sound Blaster	Creative Labs	Да

Вызовем Мастер анализа таблиц и выполним с его помощью распределение полей исходной таблицы по новым таблицам. Предлагаемая схема (модель) данных показана на рис. 10.3.

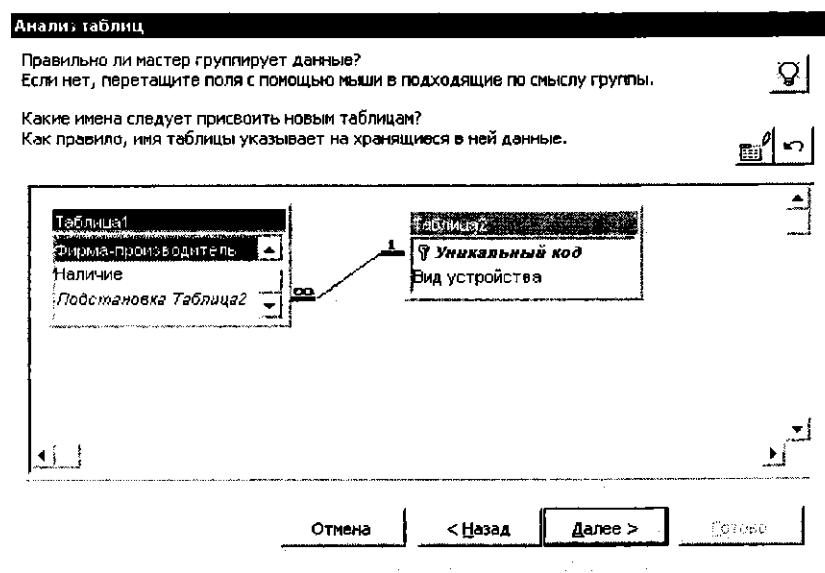


Рис. 10.3. Модель данных в окне Мастера анализа таблиц

Созданные таблицы имеют следующий вид:

Таблица1

Код	Вид устройства
а	CD-ROM
б	CD-Recorder
в	Sound Blaster

Таблица2

Фирма-производитель	Наличие	Подстановка
Acer	да	а, CD-ROM
Mitsumi	нет	а, CD-ROM
NEC	да	а, CD-ROM
Panasonic	да	а, CD-ROM
Sony	да	а, CD-ROM
Philips	нет	б, CD-Recorder
Sony	нет	б, CD-Recorder
Yamaha	да	б, CD-Recorder
Creative Labs	да	в, Sound Blaster

Сравнение содержимого этих таблиц с таблицами О2 и Д2 показывает, что Таблица1 и таблица О2 совпадают. Отличие таблиц Д2 и Таблица2 состоит в том, что пара ключевых полей («Код», «Фирма-производитель») названа новым полем «Подстановка». Поле «Подстановка» выполняет такую же роль, как и два ключевых поля.

Кроме того, Мастер анализа таблиц предлагает пользователю создать запрос, результатом выполнения которого оказывается исходная таблица. Для нашего примера Мастер создал запрос с именем «Устройства-Производители», а исходную таблицу переименовал в таблицу «Устройства-Производители_СТАРАЯ». Таким образом, поставленная задача решена.

10.4. Создание основных элементов БД

К основным элементам базы данных можно отнести таблицы, запросы, формы, отчеты, макросы и модули.

Создание базы данных

Access предоставляет два способа создания базы данных: создание пустой БД (в последующем можно добавить нужные объекты) и создание непустой БД с помощью Мастера. Первый способ отличается большей гибкостью и трудоемкостью, так как требует отдельного определения каждого элемента базы данных. Второй способ ускоряет процесс создания БД и позволяет получить базу данных с образцами информации в таблицах. Он применим в случаях, когда пользователю подходит одна из предлагаемых типовых баз

данных. Независимо от способа создания базы данных можно в любое время легко ее изменить и расширить.

При создании БД возможны несколько вариантов диалога. К примеру, сразу после запуска Access можно воспользоваться расположенной в правой части окна панелью задач, из которой создание новой БД инициируется щелчком мыши на ссылке «Новая база данных». В появившемся диалоговом окне **Файл новой базы данных (File New Database)** (рис. 10.4) нужно выбрать папку размещения базы, ввести имя и нажать кнопку **Создать (Create)**.

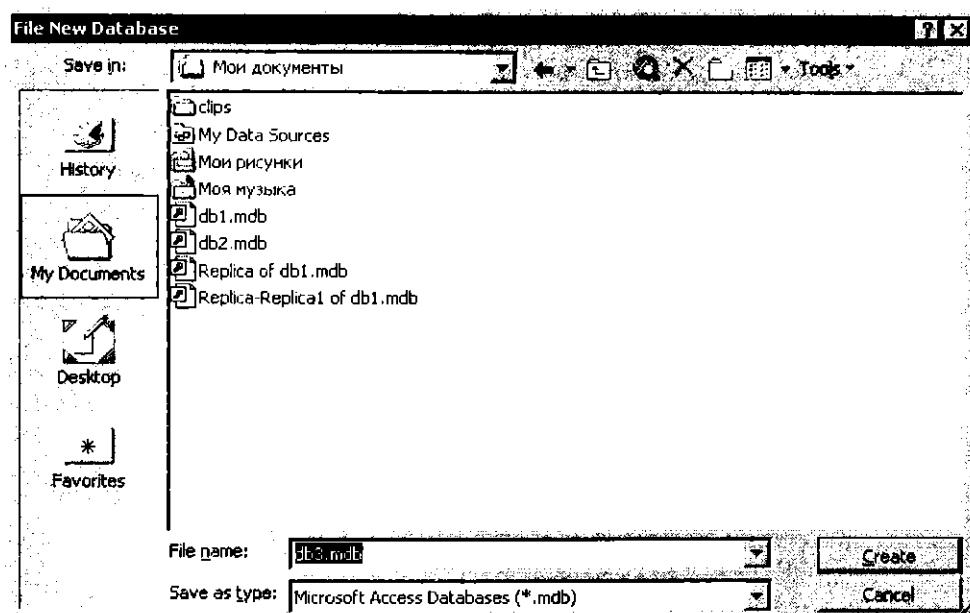


Рис. 10.4. Диалоговое окно Файл новой базы данных

В результате появляется окно открытой БД (рис. 10.5), в котором выполняется работа по созданию требуемых элементов базы данных.

Если из начального окна (рис. 10.1) пойти по ссылке «Создание с помощью шаблона» из раздела «Создание с помощью шаблона», то появится окно шаблонов с вкладками «Общие» и «Базы данных». Щелчком по соответствующей пиктограмме из вкладки «Общие» производится переход к созданию новой базы данных, для чего нужно будет задать папку и имя базы данных в окне **Файл новой базы данных**.

На вкладке **Базы данных (Databases)** можно выбрать как основу одну из многих готовых баз данных (здесь представлены различные темы, например, «Контакты», «Мероприятия», «Склад» и т. д.). Дальнейшее определение основных параметров базы данных выполняется с помощью **Мастера баз дан-**

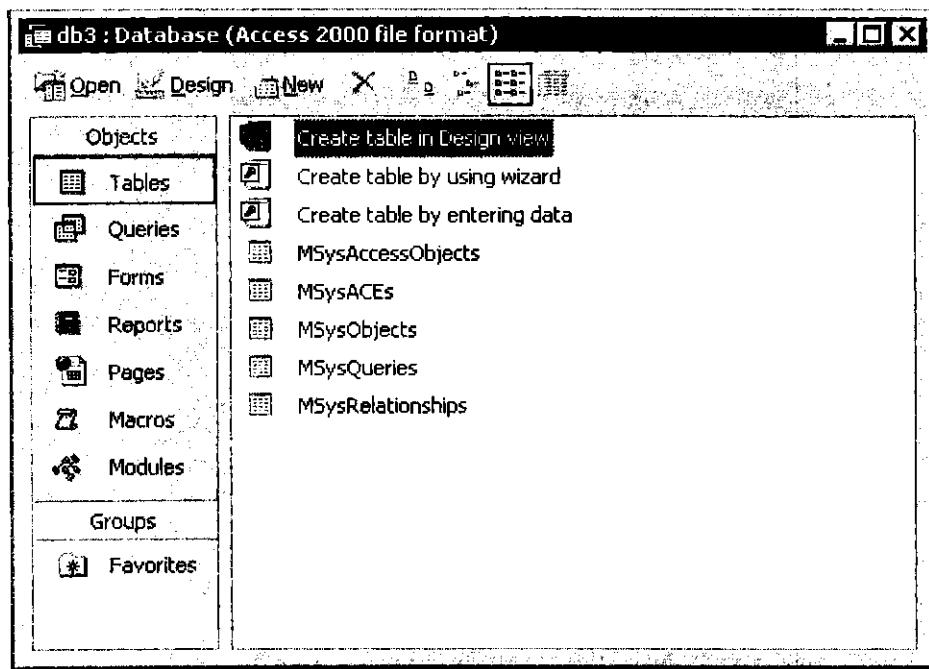


Рис. 10.5. Окно базы данных

ных. При этом можно уточнить структуру одной или нескольких таблиц (в зависимости от типа выбранной базы), вид оформления экрана, свойства отчета для печати и т. д.

Созданная БД может быть автоматически открыта и снабжена справочным окном Access с изложением понятий по объектам базы данных. Открытые и созданные Мастером базы данных имеют главные кнопочные формы, позволяющие перейти к работе с данными (окно базы данных при этом свернуто).

К созданию БД можно также перейти путем нажатия кнопки **Создать** (Create) на панели инструментов или по команде меню **Файл | Создать базу данных** (File | New Database).

Создание таблиц

Перед созданием таблицы нужно открыть базу данных, в которой таблица будет находиться. Это можно сделать с помощью начального окна (рис. 10.1), а также нажатием кнопки **Открыть базу данных** (Open An Existing Database) панели инструментов или по команде меню **Файл | Открыть** (File | Open).

В открытой БД следует выбрать вкладку Таблицы (Tables) и нажать кнопку **Создать** (New). Начать создание таблицы можно также путем выбора в

пункте Вставка (Insert) главного меню Access подиункта Таблица (Tables). В результате появляется окно, показанное на рис. 10.6.

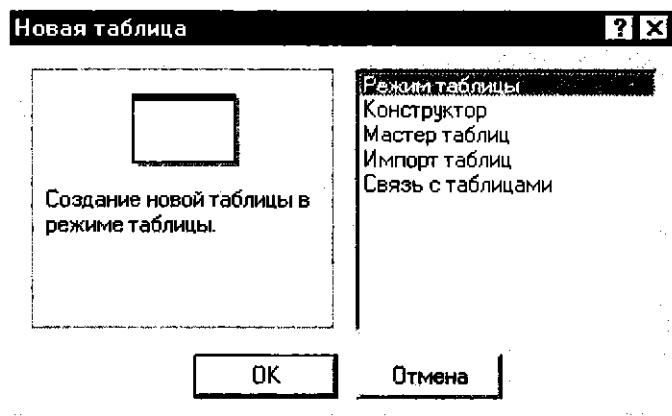


Рис. 10.6. Окно создания таблиц

Как следует из содержимого окна, возможны следующие пять вариантов создания таблиц:

- путем ввода данных в пустую таблицу, при сохранении данных в которой Access анализирует данные и автоматически присваивает соответствующий тип данных и формат каждому полю — *Режим таблицы* (*Datasheet View*);
- с помощью Конструктора — *Конструктор* (*Design View*);
- с помощью Мастера — *Мастер таблиц* (*Table Wizard*);
- из импортируемых таблиц — *Импорт таблиц* (*Import Tables*);
- путем создания таблиц, связанных с таблицами, находящимися во внешнем файле — *Связь с таблицами* (*Link Tables*).

Новые таблицы путем импортирования создаются пустыми или с данными. Это определяется параметрами (кнопка Параметры (Options)) в окне Импорт объектов (Import Objects), которое появляется при выборе исходной БД. При создании таблиц в режиме *Связь с таблицами* в окне БД появляются таблицы, которые находятся в другой базе данных. Фактически такие таблицы становятся разделяемым ресурсом, и их содержимое может изменяться из двух баз данных.

Независимо от способа создания изменение структуры таблицы можно выполнить в режиме *Конструктор*, предоставляющем наиболее гибкие и мощные возможности по определению параметров создаваемой таблицы.

Для перехода в режим *Конструктор* достаточно выделить вторую строку области выбора окна (рис. 10.6) и щелкнуть на кнопке OK. Появится окно

(рис. 10.7), предназначенное для ввода характеристик создаваемых полей, в центре которого находится специальная форма (табличного вида) описания полей. Каждому полю в этой форме соответствует одна запись.

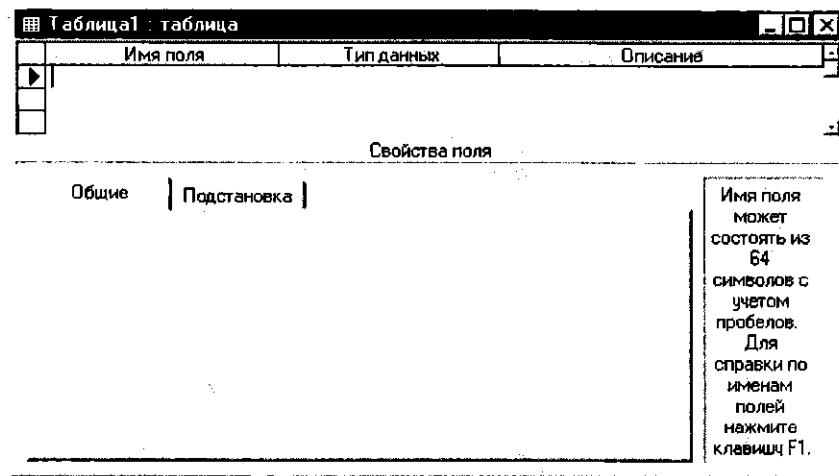


Рис. 10.7. Окно описания полей таблицы

Имена полей вводятся в первой колонке, типы полей — во второй колонке, а необязательные комментарии — в третьей. Среди всего многообразия типов полей особый интерес представляет сравнительно недавно появившийся в современных СУБД новый тип полей — *гиперссылки*. Более подробно этот тип поля рассмотрен в подразделе 10.5. При желании воспользоваться готовыми структурами таблиц можно прибегнуть к помощи Построителя таблиц, вызываемого при нажатии на кнопку панели инструментов. При этом появляется окно (рис. 10.8), позволяющее выбрать готовые описания полей.

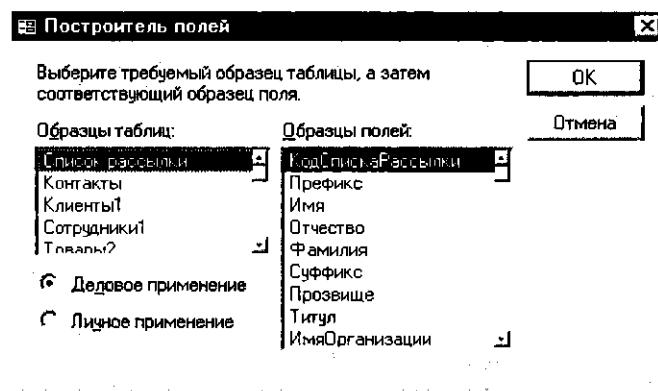


Рис. 10.8. Окно Постройтеля полей

При описании структуры таблицы следует обратить внимание на свойство полей, называемое *Индексированное поле (Indexed)*. Оно может принимать следующие значения: *Нет (No)* — не индексированное, *Да (Допускаются совпадения) (Yes (Duplicates OK))* и *Да (Совпадения не допускаются) (Yes (No Duplicates))*. Индексация поля в системе Access еще не означает, что поле является ключевым. Чтобы сделать поле ключевым, нужно сначала задать свойства поля (полноценно ключевым оно станет только при индексации этого поля, не допускающей совпадения), затем выделить строку описания поля и нажать кнопку на панели инструментов.

Считается нормой, когда таблица имеет хотя бы одно ключевое поле. Поэтому при завершении создания таблицы (например, выбором пункта меню **Файл | Сохранить (File | Save)**, нажатием кнопки **Сохранить (Save)** на панели инструментов или закрытием окна создания таблицы) при отсутствии в ней ключа Access предупреждает об этом и предлагает создать ключ типа *Счетчик (AutoNumber)*. При необходимости ключи могут быть созданы и позже.

После создания совокупности таблиц требуется выполнить связывание таблиц. Как известно, установление табличных связей позволяет контролировать целостность и достоверность информации в базе данных. Кроме того, чем раньше образованы связи между таблицами, тем удобнее создавать многотабличные запросы, формы и отчеты. При наличии связей между таблицами разработка перечисленных объектов БД ведется с учетом установленных связей.

Связывание таблиц

Для связывания таблиц БД нужно вызвать окно схемы данных. Примерный вид окна приведен на рис. 10.9.

После открытия основного окна БД окно схемы базы данных можно вызвать по команде **Сервис | Схема данных (Tools | Relationships)** основного меню или с помощью одноименной команды контекстного меню окна БД.

Построение схемы данных состоит в добавлении в нее таблиц и установлении связей между таблицами. Добавление таблиц в схему данных обычно производится с помощью соответствующей кнопки на панели инструментов.

В случае совпадения имен и типов полей у нескольких таблиц схемы данных и добавления в схему всех нужных таблиц, Access образует связи автоматически. Если образованные связи не устраивают пользователя, то их можно изменить.

Если система Access построила не все нужные связи (в общем случае, имена связываемых полей могут не совпадать) или не строила их вовсе, то образовать новую связь легко. Для этого достаточно в окне схемы данных по каждому связываемому полю таблицы выполнить следующее: выделить поле, при нажатой левой кнопке мыши протащить указатель к полю связи другой таблицы и отпустить кнопку. При этом Access предлагает определить вид и параметры связи в соответствующем диалоговом окне (рис. 10.10).

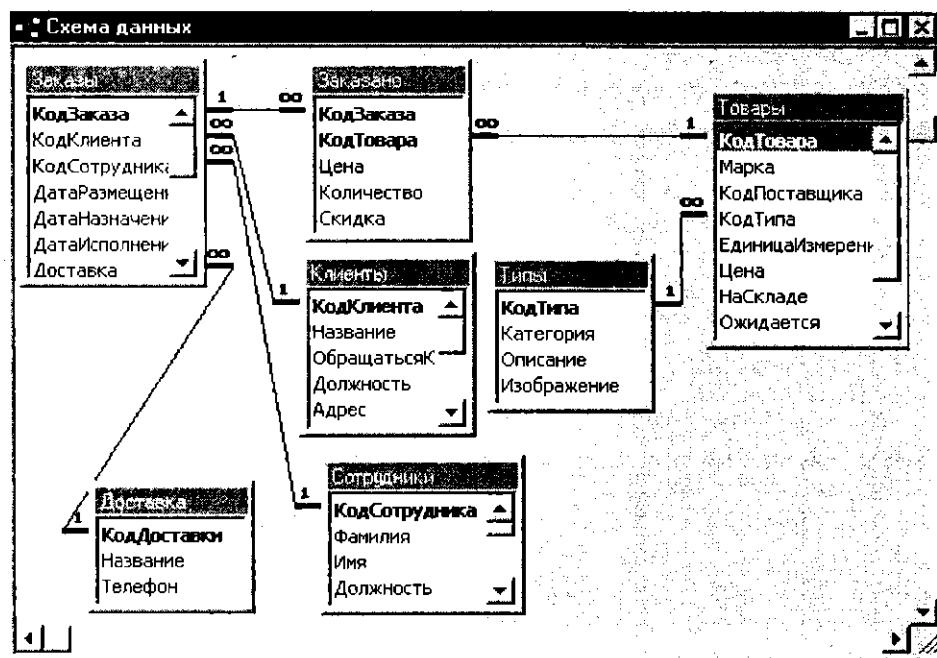


Рис. 10.9. Окно схемы базы данных

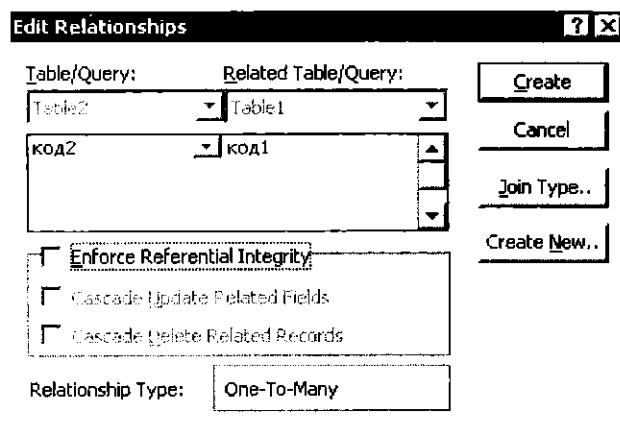


Рис. 10.10. Окно определения вида и параметров связи

Кнопка **Объединение (Join Type)** в правой части окна позволяет установить свойства объединения записей двух таблиц — как участвуют в связи остальные записи таблиц. В появившемся после нажатия этой кнопки окне для выбора предлагаются следующие варианты:

- объединение записей, в которых связанные поля таблиц совпадают;
- объединение всех записей из дополнительной таблицы и тех записей основной таблицы, в которых связанные поля совпадают;
- объединение всех записей из основной таблицы и тех записей дополнительной таблицы, в которых связанные поля совпадают.

Если установление связей производится для непустых таблиц, то Access подвергает анализу всю имеющуюся информацию, и при наличии нарушений целостности сообщает об этом.

Для изменения или удаления имеющихся связей достаточно в схеме данных подвести указатель мыши к нужной связи, выделить ее щелчком и нажать правую кнопку. Появится контекстное меню, состоящее из двух пунктов, предлагающих соответственно изменить или удалить связь. При выборе первого пункта появляется окно, показанное на рис. 10.11. Удалить связь можно также, выделив ее мышью, а затем нажав клавишу .

Система Access при управлении связыванием таблиц отличается наглядностью отображения информации о связях (рис. 10.9). Надписи, стрелки и утолщения на стрелках связей характеризуют вид связи (1:1 или 1:M), вид объединения записей (стрелка указывается при объединении записей, в которых связанные поля совпадают), а также признак контроля целостности (утолщение окончаний связывающих линий).

Создание запросов

Перед созданием запроса нужно открыть базу данных, в которой он будет храниться. После этого следует выбрать вкладку **Запросы (Queries)** и нажать кнопку **Создать (New)**. Начать создание запроса можно также, выбрав в пункте Вставка (Insert) главного меню Access подпункт **Запрос (Query)**. В результате появляется окно (рис. 10.11), в котором предлагается выбор варианта.

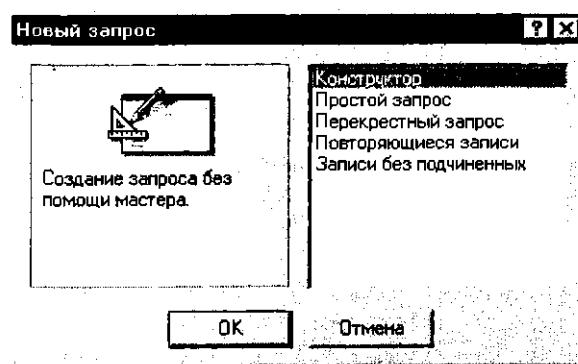


Рис. 10.11. Окно создания запроса

Возможны следующие варианты создания запроса:

- вручную с помощью конструктора — Конструктор (Design View);
- автоматическое создание простого запроса на выборку — Простой запрос (Simple Query Wizard);
- автоматическое создание перекрестного запроса для компактного представления данных в виде сводной (перекрестной) таблицы — Перекрестный запрос (Crosstab Query Wizard);
- автоматическое создание запроса на поиск записей с повторяющимися значениями полей — Повторяющиеся записи (Find Duplicates Query Wizard);
- автоматическое создание запроса на поиск записей в одной таблице, которые не имеют подчиненных записей в другой таблице — Записи без подчиненных (Find Unmatched Query Wizard).

Создаваемые запросы основаны на полях таблиц и/или запросов из базы данных. Все способы, кроме первого, реализуются с помощью Мастеров, упрощающих разработку запроса. Если созданный запрос не удовлетворяет требованиям, то можно воспользоваться Конструктором, либо создать заготовку запроса с помощью Мастера, которую затем подправить в режиме Конструктора.

Рассмотрим режим Конструктора как наиболее мощный и незаменимый при создании запросов, выходящих за рамки предлагаемых простейших вариантов.

Вызов Конструктора запросов производится при создании запроса или открытии существующего запроса и переводе его в режим Конструктора. В первом случае перейти в режим Конструктора запросов можно в окне открытой БД из вкладки Запросы (Queries) двумя способами: нажатием кнопки Создать (New) и нажатием кнопки ОК в появившемся окне (рис. 10.11), либо нажатием кнопки Конструктор (Design). Для перевода запроса в режим Конструктора достаточно щелкнуть мышью по кнопке на панели инструментов. Заметим, что при вызове Конструктора в главном меню Access появляется дополнительный пункт Запрос (Query), который имеет подпункты, позволяющие выполнять различные операции в процессе создания запроса: выполнение запроса, добавление таблицы в модель запроса, изменение вида запроса и т. д.

Составление запроса в режиме Конструктора в общем случае включает в себя определение следующего:

- таблиц и полей таблиц;
- вида запроса (выборка, добавление, удаление, перекрестный запрос, SQL-запрос);
- условий отбора записей;
- параметров отображения результатов выполнения запроса (показ полей, сортировка значений).

Все эти действия выполняются в запросной форме, которую можно отнести к форме запроса на языке QBE (подраздел 3.8). Запросная форма включает три основных элемента: заголовок (имя и тип запроса); область таблиц, их полей и связей между таблицами; бланк запроса по образцу.

Для *указания таблиц*, используемых в запросе, нужно поместить в запросную форму схемы этих таблиц или запросов и указать связи между ними. Включение объектов в запрос производится в окне **Добавление таблицы** (**Show Table**), которое вызывается автоматически (при создании запроса) или принудительно при работе с запросом путем нажатия кнопки панели инструментов.

При создании запроса Access по умолчанию предоставляет заготовку запроса на выборку. Изменить вид запроса можно с помощью пункта **Запрос** (**Query**) основного меню системы, где возможные виды запросов (выборка, обновление, добавление, удаление, создание таблицы, перекрестный) перечислены как подпункты меню.

Для создания запроса в виде инструкции языка SQL можно при наличии на экране запросной формы воспользоваться пунктом меню **Запрос | Запрос SQL** (**Query | SQL**), в котором выбрать нужный вид SQL-запроса (на объединение, к серверу или управляющий). Подробнее об использовании языка SQL в Microsoft Access речь идет в подразделе 10.6.

Пример запроса на выборку из двух таблиц, связанных связью 1:1 по полям «поле12» и «поле21» таблиц 1 и 2 соответственно, с отображением полей «поле11», «поле12» и «поле23» в результирующей таблице приведен на рис. 10.12.

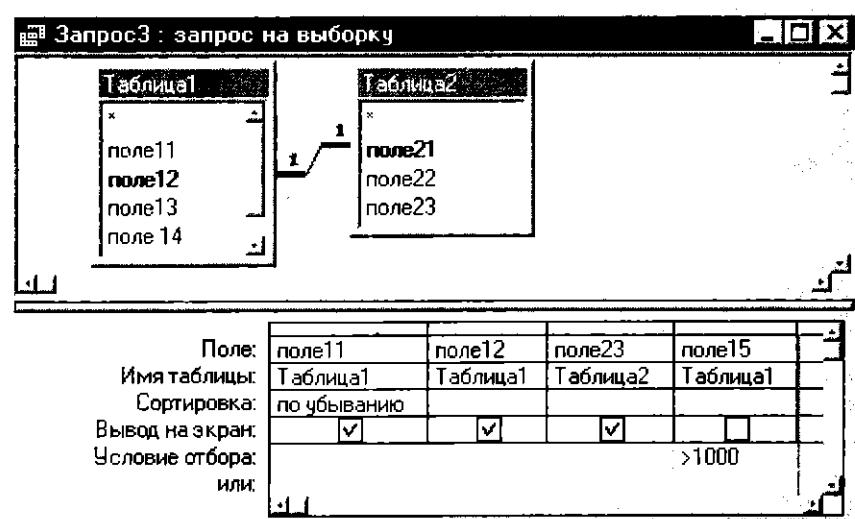


Рис. 10.12. Запрос на выборку

Условием отбора записей для вывода является выражение «>1000», означающее, что значения в поле «поле15» должны быть больше 1000 (само поле не выводится). Выводимые записи сортируются по убыванию значений в поле «поле11».

Завершить создание запроса можно следующим образом:

- выбором пункта меню Файл | Сохранить (File | Save);
- выбором пункта меню Файл | Сохранить как/Экспорт (File | Save As/ Export);
- нажатием кнопки Сохранить (Save) на панели инструментов;
- путем закрытия окна создания запроса.

Создание форм

Перед началом создания формы следует открыть базу данных, в которой она будет находиться. Собственно создание формы производится в окне открытой БД путем выбора вкладки **Формы (Forms)** и нажатия кнопки **Создать (New)**. Начать создание формы можно также, выбрав в пункте **Вставка (Insert)** главного меню Access подпункт **Форма (Form)**. При этом появляется окно **Новая форма** (рис. 10.13).

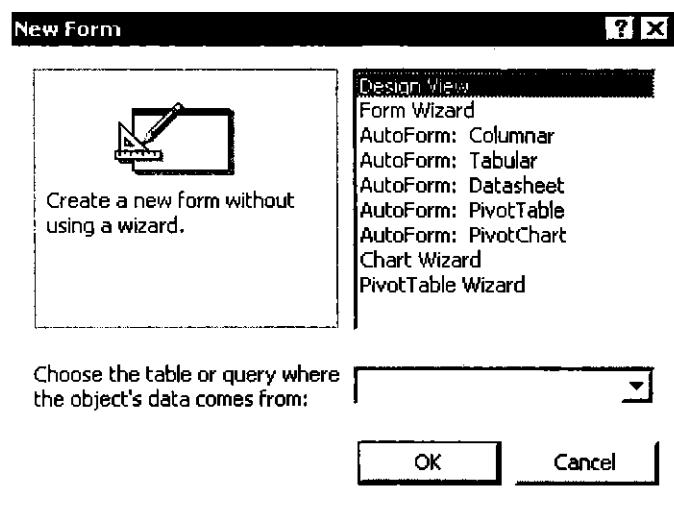


Рис. 10.13. Окно создания формы

Возможны 9 вариантов создания формы:

- 1) с помощью Конструктора – Конструктор (Design View);
- 2) с помощью Мастера – Мастер форм (Form Wizard);
- 3) автоматическое создание формы стандартного вида, в которой поля размещены в столбец – Автоформа: в столбец (Autoform: Columnar);

- 4) автоматическое создание стандартной формы в виде таблицы — **Автоформа: табличная (Autoform: Tabular)**;
- 5) автоматическое создание стандартной формы, незначительно отличающейся по виду от табличной формы — **Автоформа: ленточная (Autoform: Datasheet)**;
- 6) автоматическое создание формы в виде сводной таблицы с помощью мастера — **Автоформа: сводная таблица**;
- 7) автоматическое создание формы в виде сводной диаграммы с помощью мастера — **Автоформа: сводная диаграмма**;
- 8) создание формы с диаграммой — **Диаграмма (Chart Wizard)**;
- 9) создание формы со сводной таблицей Microsoft Excel — **Сводная таблица (PivotTable Wizard)**.

Наиболее просто создать форму по вариантам 3–5. В этих случаях получаются несложные формы, включающие все поля источника данных (таблицы или запроса). После создания форма доступна для просмотра/редактирования данных.

Запустив Мастер (вариант 2), пользователь может создать формы таких же видов, как и при выборе вариантов 3–5. Но здесь можно выбрать в качестве источников данных произвольное число запросов и/или таблиц, включив в форму нужные поля. Кроме того, можно изменить стиль фонового изображения. При необходимости произвести другие изменения макета формы, после завершения работы с Мастером, можно перейти в режим Конструктора.

Режим Конструктора является наиболее мощным, но и более трудоемким средством разработки форм. Чтобы сократить общее время разработки, целесообразно перед вызовом Конструктора воспользоваться одним из других способов создания формы, а Конструктор использовать для окончательного приведения формы к нужному виду. Рассмотрим основные возможности Конструктора.

Вызов Конструктора форм производится при создании формы или открытии существующей формы и переводе ее в режим Конструктора. В первом случае перейти в режим Конструктора форм можно в окне открытой БД из вкладки **Формы (Forms)** путем нажатия кнопки **Создать (New)** и нажатия **OK** в появившемся окне (рис. 10.13), либо нажатием кнопки **Конструктор (Design)**. Для перевода открытой формы в режим Конструктора достаточно щелкнуть мышью по кнопке на панели инструментов.

Окно разработки формы выглядит, как показано на рис. 10.14. Форма в режиме Конструктора в общем случае содержит следующие области: заголовок и примечание формы, верхний и нижний колонтитулы и область данных (в которой отображаются данные источников). Обязательной является область данных, остальные — необязательные. В области данных обычно размещаются поля таблиц.

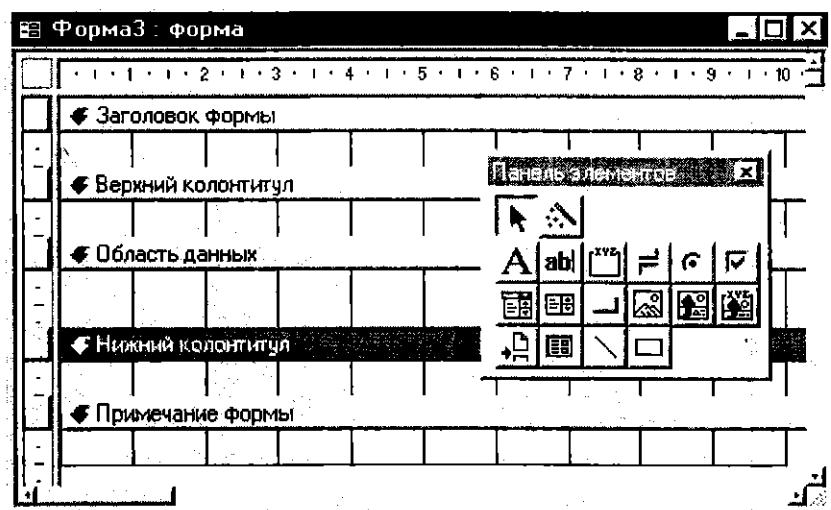


Рис. 10.14. Окно формы в режиме Конструктора

Управлять отображением необязательных областей можно с помощью пункта меню **Вид (View)** при открытой форме в режиме Конструктора. Основные возможности Конструктора определяются составом инструментов Панели элементов (Toolbox), показанной на рис. 10.14. Панель элементов содержит следующие основные инструменты (перечислены слева направо и сверху вниз):

- **Выбор объектов (Select Objects)** — используется для выделения, изменения размера, перемещения и редактирования элементов управления.
- **Мастера (Control Wizards)** — кнопка включения/отключения автоматического вызова Мастеров создания элементов управления. Существуют Мастера по созданию группы, поля со списком, списка и командной кнопки.
- **Надпись (Label)** — предназначена для создания элемента управления, содержащего неизменяемый текст. По умолчанию большинство элементов управления содержит присоединенный текстовый элемент.
- **Поле (Text Box)** — используется для отображения, ввода или изменения данных, содержащихся в источнике записей, вывода результатов вычислений, а также приема данных, вводимых пользователем.
- **Группа переключателей (Option Group)** — служит для создания группы элементов (флажков, переключателей или выключателей), представляющих набор альтернативных значений, из которых выбирается одно значение. Если группа присоединена к полю базового запроса, на котором основана форма, или таблицы, то при выборе одного из элементов группы его значение присваивается полю.

- **Выключатель (Toggle Button), Переключатель (Option Button) и Флажок (Check Box)** – различные по виду, но одинаковые по использованию элементы – предназначены для отображения логических значений. Выбор (включение) элемента приводит к вводу в соответствующее логическое поле значения «Да», «Истина» или «Вкл» (определяется значением свойства поля «Формат поля»). Повторный выбор элемента изменяет значение на противоположное: «Нет», «Ложь» или «Выкл». Эти элементы можно помещать в группу. Вид отображаемого в базовой таблице значения зависит от свойства Тип элемента управления (вкладка Подстановка свойств поля).
- **Поле со списком (Combo Box)** представляет составной элемент управления, объединяющий поле и раскрывающийся список. Для ввода значения в поле базовой таблицы, можно ввести значение в поле или выбрать значение из списка.
- **Список (List Box)** – предназначен для создания перечня (списка) возможных значений. Список можно создать, явно вводя данные, либо указав источник данных – таблицу или запрос. Список может содержать несколько столбцов, причем установка ширины при отображении любого из них делает этот столбец скрытым. Полю, с которым связан список, присваивается значение из множества значений одного любого столбца, в том числе скрытого.
- **Кнопка (Command Button)** – обычно используется для запуска закрепленного за ней макроса или программы на языке Visual Basic.
- **Рисунок (Image)** – предназначен для размещения в форме неизменяющегося рисунка.
- **Свободная рамка объекта (Unbound Object Frame)** – используется для размещения объекта из приложения, поддерживающего технологию OLE. Включаемый объект становится частью формы, но не хранится в таблице БД. В качестве объекта может быть электронная таблица, рисунок, диаграмма, звуковой файл и т. д.
- **Присоединенная рамка объекта (Bound Object Frame)** – используется для включения в форму OLE-объектов. Некоторые объекты могут отображаться в форме (например, рисунки, диаграммы или электронные таблицы), другие – выводят в форме в виде значка приложения, в котором этот объект был создан.
- **Разрыв страницы (Page Break)** – позволяет вставлять разрыв страницы в многостраничной форме.
- **Набор вкладок (Tab Control)** – позволяет создать в форме несколько вкладок, каждая из которых может содержать другие элементы управления.
- **Подчиненная форма/отчет (Subform/Subreport)** – предназначен для внедрения в форму некоторой другой (подчиненной) формы.

- **Линия (Line) и Прямоугольник (Rectangle)** — предназначены для создания соответствующих геометрических фигур.
- **Другие элементы (More Controls)** — кнопка, открывающая список всех установленных в системе элементов управления ActiveX.

Замечание.

При разработке отчетов в режиме Конструктора предоставляется аналогичная панель элементов. Отсюда — унифицированное название отдельных инструментов (например, Подчиненная форма/отчет), а также текст справки, описывающий работу с формами и с отчетами. Более того, Access позволяет форму преобразовать в отчет (см. справочную систему в разделе «Формы» на странице «Сохранение формы в виде отчета»).

Создание отчетов

Перед созданием отчета нужно открыть базу данных, в которой он будет находиться. Собственно создание отчета производится путем нажатия кнопки Создать (New) в окне открытой БД на вкладке Отчеты (Reports). Начать создание отчета можно также, задав в пункте Вставка (Insert) главного меню Access команду Отчет (Report). В результате появляется окно, показанное на рис. 10.15.

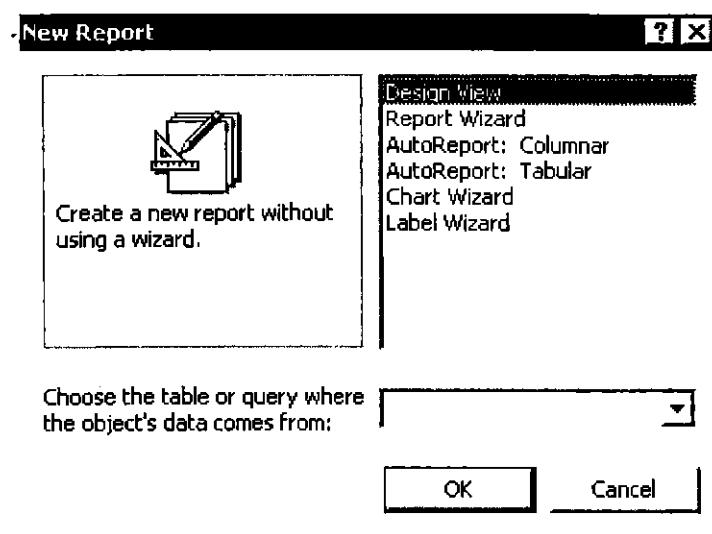


Рис. 10.15. Окно создания отчета

- Возможны следующие варианты создания отчета:
- 1) с помощью Конструктора — Конструктор (Design View);
 - 2) с помощью Мастера — Мастер отчетов (Report Wizard);
 - 3) автоматическое создание отчета стандартного вида, в котором каждая запись базового запроса или таблицы представлена в виде названия и значения поля — Автоотчет: в столбец (Autoreport:Columnar);
 - 4) автоматическое создание стандартного отчета, в котором данные записи базового запроса или таблицы выводятся в одной строке — Автоотчет: ленточный (Autoreport:Tabular);
 - 5) создание отчета с диаграммой — Диаграмма (Chart Wizard);
 - 6) создание отчета для печати почтовых наклеек — Почтовые наклейки (Label Wizard).

Проще всего создать отчет по вариантам 3 и 4, требующим наименьшее число параметров. В этих случаях получаются простейшие отчеты, включающие все поля источника данных (таблицы или запроса). После создания отчет доступен для просмотра или печати. В режиме Конструктора предоставляются более мощные средства, требующие больше знаний и времени для разработки отчета.

Техника работы с Конструктором отчетов мало чем отличается от работы с Конструктором форм. В частности, при этом используется такая же панель элементов (рис. 10.14). Остановимся на важнейших особенностях разработки отчетов в сравнении с формами.

При работе над отчетом используются те же области, что и при создании форм: области заголовка и примечания, области верхнего и нижнего колонтитулов, а также область данных. Кроме того, в отчеты можно включать области группировки записей.

Заголовок отчета и примечание выводятся один раз: в начале и конце отчета соответственно. Верхний/нижний колонтитулы помещаются в начало/конец каждой страницы отчета. Содержимое области данных выводится один раз для каждой записи исходной таблицы или запроса. Если пользователь задал группировку записей отчета, то по каждому полю, по которому проводится группировка данных, Access формирует заголовок и примечание группы. Для создания в отчете области группировки нужно при открытом в режиме Конструктора отчете выбрать пункт меню Вид/Сортировка и группировка (View/Sorting And Grouping) или нажать кнопку  на панели инструментов.

Существенное различие между отчетом и формой заключается в том, что отчеты предназначены исключительно для вывода данных на печать. Поэтому в них можно отказаться от использования (доступных для включения в отчет) управляющих элементов для ввода данных: списков, полей со списком, переключателей и т. п.

Отчеты могут находиться в двух режимах: Конструктора или Просмотра. Просмотреть готовый отчет можно после выделения нужного отчета при нажатии кнопки Просмотр (Preview) в окне БД, выборе пункта меню Файл/Предварительный просмотр (File/Preview) или нажатии кнопки  на панели инструментов.

Создание макросов

Перед созданием макроса нужно открыть базу данных, в которой он будет находиться. Собственно создание макроса производится в окне открытой БД путем выбора вкладки Макросы (Macros) и нажатия кнопки Создать (New). Начать создание макроса можно также, выбрав в пункте Вставка (Insert) главного меню Access подпункт Макрос (Macros). В результате открывается окно создания макроса (рис. 10.16).

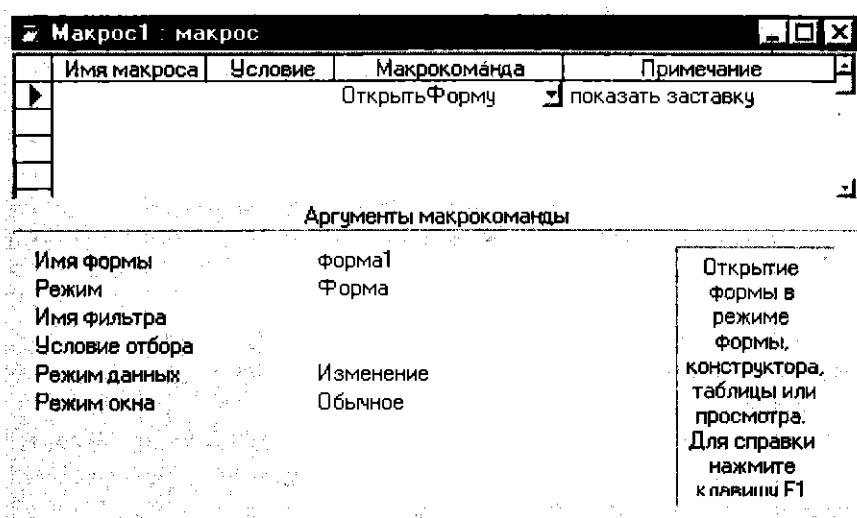


Рис. 10.16. Окно создания макроса

Окно создания макроса в общем случае содержит следующие колонки: Имя макроса (Macro Names), Условие (Conditions), Макрокоманда (Action) и Примечание (Comment). Первые две колонки не являются обязательными при создании макросов, поэтому их можно убрать с помощью пункта Вид (View) меню Access или соответствующих кнопок:  и  панели инструментов.

Каждый макрос включает в себя одну или несколько макрокоманд, которые могут выполняться безусловно или в соответствии с некоторым условием. В последнем случае напротив макрокоманды указывают условное выражение.

жение или многоточие «...». Если логическое выражение в строке макрокоманды истинно, то выполняется эта макрокоманда и все последующие, в поле **Условие (Conditions)** которых стоит многоточие. В случае ложности логического выражения пропускается текущая макрокоманда и все непосредственно следующие за ней макрокоманды, содержащие многоточие в поле **Условие (Conditions)**.

Для удобства хранения связанные по смыслу макросы можно объединять в группы. Полное имя макроса из группы включает в себя имя группы и отделенное от него точкой имя макроса. *Макрокоманды* в макросах представляют собой операции с параметрами из фиксированного в СУБД списка.

Все множество макрокоманд Access по функциональному принципу можно условно разделить на следующие группы:

- открытие и закрытие таблиц, форм и отчетов;
- вывод данных;
- выполнение запроса;
- проверка истинности условий и управление выполнением макрокоманд;
- установка значений;
- поиск данных;
- построение специального меню и выполнение команд меню;
- управление выводом на экран и фокусом;
- сообщение пользователю о выполняемых действиях;
- переименование, копирование, удаление, импорт и экспорт объектов;
- запуск других приложений.

Ввод макрокоманд пользователем максимально облегчен, поскольку названия самих макрокоманд, а также значения многих аргументов можно не только вводить с клавиатуры, но и выбирать из списка (в ячейке столбца **Макрокоманда (Action)** или поля аргумента макрокоманды). Выражения в области аргументов и условий выполнения макрокоманд можно непосредственно вводить с клавиатуры или использовать Построитель выражений. Процесс ввода (выбора из списка) макрокоманд и их аргументов сопровождается подсказкой в правой нижней части окна создания макроса (рис. 10.17). Это существенно облегчает ввод макрокоманд. Для добавления в макрос других макрокоманд нужно в области ввода макрокоманд перейти на следующую строку. Макрокоманды выполняются в порядке их расположения в бланке.

Замечание.

Существует прием быстрого создания макроса, выполняющего действия над конкретным объектом. Он состоит в выборе объекта (таблицы, формы, макроса и пр.) в окне базы данных и перемещении его с помощью мыши в ячейку макрокоманды в окне макроса. При переносе макроса в ячейку макро-

команды вводится макрокоманда, запускающая этот макрос, а при переносе других объектов — в макрос добавляется макрокоманда открытия объекта.

Завершая создание макроса, требуется задать его имя. Макрос с именем AutoExec запускается автоматически при открытии базы данных. Временно отменить автоматический запуск этого макроса можно с помощью удержания клавиши <Shift> в момент открытия БД. Используя возможности автозапуска макроса, удобно выполнять различные подготовительные операции над БД после ее открытия. К таким операциям относятся вывод заставки (например, формы с картинкой) и открытие главной управляющей кнопочной формы.

Созданные и хранимые в БД макросы могут запускаться пользователем либо вызываться из других макросов или программ на Visual Basic, а также при возникновении определенных событий в БД. **Событие** — это любое распознаваемое объектом действие, на которое можно задать реакцию. События возникают в результате действий пользователя, выполнения инструкций Visual Basic или генерируются системой Access. Типичными событиями в Microsoft Access являются, например, нажатие кнопки мыши, изменение данных, а также открытие/закрытие формы или отчета.

Автоматический вызов макросов производят в случае наступления событий в таких объектах БД, как формы и отчеты. Всего существует около 40 событий. По функциональному назначению события можно разделить на следующие группы:

- события данных (Data Events) возникают при вводе, удалении или изменении данных в форме или элементе управления, а также при перемещении фокуса с одной записи на другую;
- события клавиатуры (Keyboard Events) возникают при вводе с клавиатуры, а также при передаче нажатий клавиш с помощью макрокоманды «КомандыКлавиатуры» (SendKeys) или инструкции SendKeys;
- события ошибки и таймера (Error and Timing Events) используются при обработке ошибок и синхронизации данных в формах;
- события мыши (Mouse Events) возникают при действиях с мышью, например при нажатии кнопки мыши или при удержании кнопки в нажатом положении;
- события фильтра (Filter Events) возникают при создании или применении фильтра в форме;
- события печати (Print Events) возникают при печати отчета или при его форматировании для печати;
- события фокуса (Focus Events) возникают, когда форма или элемент управления теряют/получают фокус, а также в момент, когда они становятся активными/неактивными;
- события окна (Window Events) возникают при открытии, изменении размеров или закрытии формы или отчета.

Для обработки событий в БД Access можно использовать макросы или процедуры обработки событий (на языке Visual Basic для приложений). Чтобы организовать обработку события, нужно в ячейке свойства этого события объекта (формы, отчета или элемента управления) ввести имя макроса или выбрать элемент [Процедура обработки событий] и нажать кнопку Построителя (рис. 10.17). В последнем случае в соответствующем окне можно написать нужную VBA-программу.

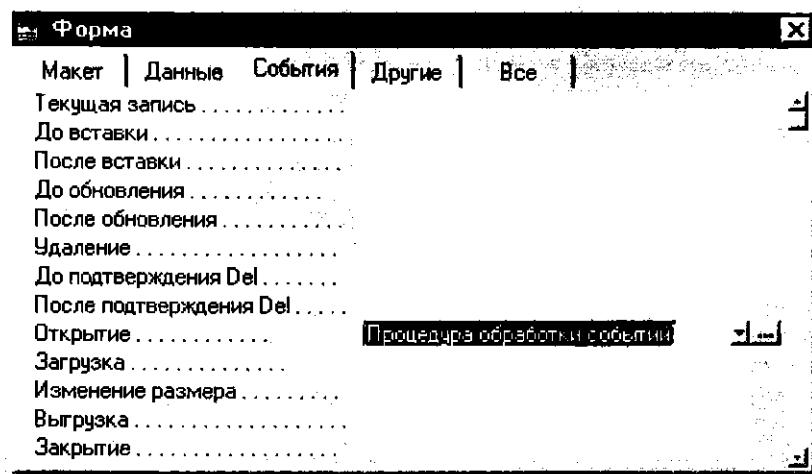


Рис. 10.17. Создание процедуры обработки события

Различные объекты БД имеют свой перечень событий, которые могут возникать в процессе работы с ними в различных режимах. Каждое действие может явиться причиной ряда других событий, возникающих в определенной последовательности. Понимание того, когда и в какой последовательности возникают события, существенно, поскольку порядок событий определяет условия и очередность выполнения макросов и процедур обработки событий.

Название свойства, как правило, происходит от имени события или даже совпадает с ним. Так, форма или отчет имеют свойство Открытие (OnOpen), позволяющее задать макрос или процедуру обработки события при возникновении события Открытие (Open). Имя макроса или процедуры обработки события является значением свойства.

Для изменения процедуры обработки события нужно выполнить следующее:

1. В режиме Конструктора формы/отчета открыть окно свойств объекта или его элемента и выбрать вкладку События (Events).
2. Выбрать свойство события, в ответ на которое должна выполняться процедура, и нажать кнопку Построителя рядом с ячейкой свойства. Откроется окно диалога Построитель (Builder).

3. Выбрать двойным щелчком кнопки мыши элемент **Программы (Programs)**. В окне модуля откроется процедура (если пользователь не создавал ее ранее — заготовка процедуры, содержащая инструкции **Sub** и **End Sub**).
4. Ввести текст программы, которая должна выполняться в ответ на события.

Создание модулей

Для автоматизации выполнения некоторых действий в простейших приложениях используют макросы. В некоторых случаях обойтись макросами не удается и приходится прибегать к написанию VBA-программ. К примеру, использование программирования может помочь в решение следующих задач:

- обработка ошибок в приложении;
 - создание собственной функции обработки, например, округления значения;
 - получение прямого доступа к функциям Windows API;
 - создание приложения с высокой производительностью (программы компилируются и выполняются быстрее, чем макросы);
 - создание новых объектов БД во время работы приложения.
- VBA-программы хранятся в **модулях** БД, которые бывают двух видов:
- стандартные — создаются пользователем и видны во вкладке БД **Модули (Modules)**;
 - модули форм/отчетов — создаются автоматически и являются частью этих объектов (форм/отчетов).

Программы на языке VBA составляются в виде процедур двух типов: функций (описатель **Function**) и подпрограмм (описатель **Sub**). Процедура может иметь параметры, через которые ей передаются значения. Основное отличие функции от подпрограммы состоит в том, что первая может в точку вызова вернуть единственное значение, вторая — значений не возвращает. Кроме того, функцию можно вызывать из различных мест, в том числе из макросов и выражений, используемых в запросах, таблицах (в условиях на значение (**Validation Rule**)) или формах. Подпрограмму можно вызвать из функции или как процедуру обработки события в форме или отчете.

Стандартные модули используются для создания и хранения процедур, вызываемых из различных объектов БД. Модули *форм/отчетов* предназначены для создания и хранения процедур, используемых в форме/отчете. В каждом модуле формы/отчета в БД содержатся заготовки процедур обработки событий, которые вызываются в ответ на события, возникающие в форме

или отчете. Пользователь может создавать собственные процедуры обработки событий, добавляя тексты VBA-программ в эти заготовки.

Для создания стандартного модуля требуется в окне открытой базы данных (рис. 10.5) выбрать вкладку **Модули (Modules)** и нажать кнопку **Создать (New)**. Начать создание модуля можно также, задав в пункте **Вставка (Insert)** главного меню Access команду **Модуль (Module)**.

Для редактирования модуля формы/отчета в окне открытой БД нужно выделить мышью соответствующий объект или перевести его в режим Конструктора. Затем в меню **Вид (View)** выбрать пункт **Программа (Code)** или нажать кнопку панели инструментов. Ввод и редактирование текстов программ стандартного модуля или модуля формы/отчета производится в окне модуля (рис. 10.18).

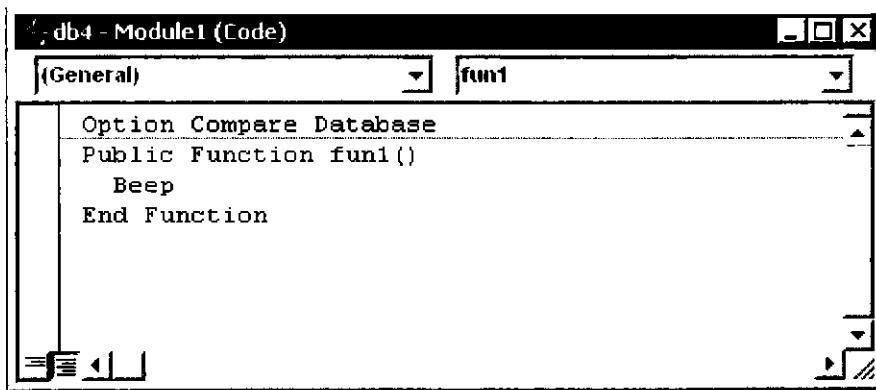


Рис. 10.18. Окно модуля

В верхней части окна модуля находятся следующие два раскрывающихся списка: список объектов (слева) и список процедур (справа).

1. Список объектов. При редактировании модуля формы/отчета в этом списке можно выбрать форму/отчет (целиком), необходимый раздел или элемент управления, способный генерировать событие. Для выбора раздела описания имеется элемент [Общая область] ([General]). Стандартный модуль в списке **Объект** содержит только элемент [Общая область].
2. Список процедур — есть список процедур выбранного объекта из первого списка, которые можно редактировать. Выбранная в этом списке процедура выводится в текстовой области окна модуля. На рис. 10.19 приведена процедура **fun1** подачи звукового сигнала.

Сохранить введенные тексты программ можно командой **Сохранить (Save)** пункта меню **Файл (File)**, с помощью кнопки сохранения панели инструментов или закрытием окна модуля с последующим подтверждением о сохранении.

Созданные и находящиеся в файле базы данных VBA-программы при первом обращении к ним автоматически компилируются Access, после чего вызываются на выполнение. Для ускорения работы приложения VBA-программы можно заранее откомпилировать и сохранить в базе данных вместе с исходными текстами. Для этого нужно открыть любой модуль в БД и выполнить команду Отладка | Компилировать (Debug | Compile). После этого, находясь в окне модуля (рис. 10.19), нужно сохранить результат компиляции с помощью команды Файл | Сохранить (File | Save). Платой за ускорение вызова VBA-программ БД является увеличение размера mdb-файла базы данных.

10.5. Работа с гиперссылками

В числе возможных типов полей таблиц Access 2002 имеется сравнительно недавно появившийся, но в то же время весьма полезный тип данных — Гиперссылка (Hyperlink). Он позволяет хранить в поле простые или сложные ссылки на файлы, документы и другие объекты, находящиеся как в базе данных, так и вне ее.

Характеристика гиперссылки

Гиперссылки могут содержать URL-адрес в сети Internet или intranet или сетевой маршрут в формате UNC к файлу на сервере локальной сети или на диске локального компьютера. Ссылка может указывать на файл в формате HTML или в формате, поддерживаемом приложением OLE или ActiveX, установленном на компьютере.

Поле гиперссылки по структуре хранимой информации является текстовым и может содержать до 2048 символов. Текст гиперссылки можно считать ее значением. Оно включает в себя до трех частей: описание (необязательное), основной адрес гиперссылки и дополнительный адрес (необязательный). Составные части отделяются друг от друга символом числа (#). Описание представляет собой текст, отображаемый в поле или элементе управления, кроме режима редактирования гиперссылки. Адрес гиперссылки — это URL- или UNC-адрес (например: <http://home.netscape.com/comprod/index.html> или <\\Serv\Market\Reclama.doc>). Дополнительный адрес задает именованный объект внутри файла (например, диапазон ячеек в рабочем листе Excel или закладка в документе Word).

По технике работы поле гиперссылки аналогично полю **Объект OLE (OLE Object)**. В операциях создания и редактирования оно имеет сходство с обычным текстовым полем. Поля гиперссылок, как и другие поля, создаются в таблицах, а затем размещаются в других объектах базы данных: формах, запросах и отчетах. Чтобы увидеть поле гиперссылки в режиме просмотра таблицы, откроем таблицу Поставщики учебной базы «Борей» и сделаем видимой ко-

лонку Основная страница (рис. 10.19). В этой колонке имеются гиперссылки — текст с подчеркиванием.

Поставщики : таблица					
Область	Индекс	Страна	Телефон	Факс	Основная страница
LA	70117	США	(100) 555-4822		Cajun.htm
MI	48104	США	(313) 555-5735	(313) 555-3349	
	100	Япония	(03) 3555-5011		
Asturias	33007	Испания	(98) 598 76 54		
	545	Япония	(06) 431-7877		Mayumi (на Web)
Victoria	3058	Австралия	(03) 444-2343	(03) 444-6588	
	M14 GSD	Великобритания	(161) 555-4448		
	S-345 67	Швеция	031-987 65 43	031-987 65 91	
	5442	Бразилия	(11) 555 4640		
	101785	Россия	(095) 998-4510		
	60439	Германия	(069) 992755		Plutzer (на Web)
	27478	Германия	(04721) 8713	(04721) 8714	
	48100	Италия	(0544) 60323	(0544) 60603	Formaggi.htm
	1320	Норвегия	(0)2-953010		
OR	97101	США	(503) 555-9931		
	6-123 45	Швейцария	08123 45 57		

Рис. 10.19. Гиперссылки в таблице

Некоторые HTML-документы, на которые имеются ссылки из поля гиперссылки таблицы Поставщики, размещаются в той же папке, что и база данных «Борей». Еще один пример использования гиперссылки в этой базе данных — кнопка Просмотр списка товаров в форме Товары.

Замечание.

В случае, если у вас установлена учебная база данных Northwind.mdb, в освоении приемов работы с гиперссылками вам поможет таблица Suppliers (поле Home Page).

Основные операции по работе с гиперссылками можно выполнить, используя контекстное меню поля гиперссылки. Для этого надо установить указатель мыши на поле гиперссылки, вызвать контекстное меню и выполнить команду Гиперссылка (Hyperlink). В число команд входят команды, позволяющие открыть ссылочный документ, копировать гиперссылку в буфер обмена, добавить гиперссылку в папку Избранное (Favorites), изменить гиперссылку или ее описание.

Создается поле гиперссылки при описании или изменении структуры таблицы БД. Для этого типа поля требуется в области указания типа поля ввести или выбрать из списка слово Гиперссылка (Hyperlink). Рассмотрим, как активизировать, вставить и редактировать значение поля гиперссылки.

Активизация гиперссылки

Активизация гиперссылки представляет собой переход по адресу, заданному выбранным значением поля гиперссылки. Для активизации гиперссылки можно подвести указатель мыши к значению поля гиперссылки и щелкнуть левой кнопкой или, выделив значение поля с помощью клавиш управления курсором (например, клавиши <Tab>) и нажать клавишу <Enter>. В результате Access запускает соответствующее приложение и передает ему основной и дополнительный адреса. Если проделать это со значением гиперссылки в верхней записи таблицы (рис. 10.19), то будет вызван Internet Explorer. После завершения операций с объектом, на который был выполнен переход, происходит возврат в точку вызова.

Вставка гиперссылки

Чтобы начать вставку гиперссылки в пустое поле, нужно, прежде всего, установить текущим поле гиперссылки нужной записи таблицы, для чего переместить курсор в это поле (клавишами управления курсором или мышью). После этого можно выполнить следующее:

- начать непосредственный ввод адреса гиперссылки, опуская описательную часть. Это простой, но не всегда удобный способ, особенно если точно адрес не известен;
- с помощью контекстного меню или команды меню Вставка | Гиперссылка (Insert | Hyperlink) вызвать диалоговое окно Добавить гиперссылку (Insert Hyperlink)(рис. 10.20).

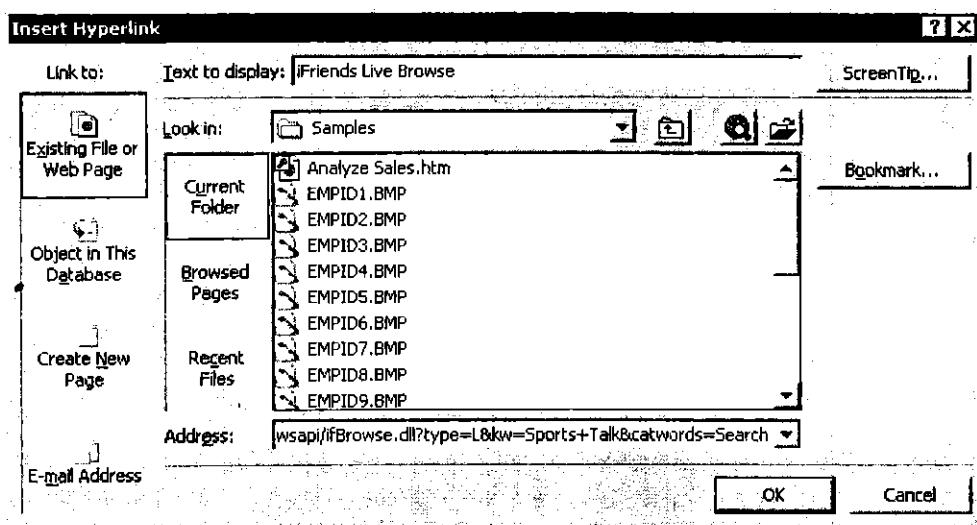


Рис. 10.20. Диалоговое окно вставки гиперссылки

В появившемся диалоговом окне с помощью списка слева следует сначала выбрать тип гиперссылки: существующий файл или страница в Интернете (Existing File or Web Page), объект этой базы данных (Object in This Database), страница доступа к данным, создаваемая тут же (Create New Page), либо адрес электронной почты (E-mail Address). После этого в центральной части окна необходимо уточнить параметры вставляемой гиперссылки в зависимости от ее типа.

Так, в первом случае это может быть имя файла в текущей папке (Current Folder). Если же ссылочным документом является база данных Access, то в качестве возможных объектов могут выступать все ее объекты: таблицы, запросы, формы, отчеты, страницы доступа к данным, макросы и модули. Сохранение создаваемой гиперссылки в таблице происходит при нажатии кнопки ОК.

Редактирование гиперссылки

Для перехода к редактированию гиперссылки можно поступить следующим образом.

1. Щелкнуть мышью на соседнем слева поле в области просмотра таблицы и нажать клавишу <Tab>. Гиперссылка будет выделена.
2. Нажать клавишу <F2>. Произойдет переход к режиму посимвольного редактирования текста гиперссылки. В поле гиперссылки появится текстовая строка, доступная для редактирования (рис. 10.21).

Поставщики : таблица			
Область	Страна	Телефон	Основная страница
	Россия	(095) 325-2222	
LA	США	(100) 555-4822	Сайт.htm
MI	США	(313) 555-5735	
Asturias	Япония	(03) 3555-5011	Таблица1#..\..\test1\db11.mdb#Table_Таблица1
	Испания	(98) 598 76 54	
	Япония	(06) 431-7877	Mayumi (на Web)
Victoria	Австралия	(03) 444-2343	
	Великобритания	(161) 555-4448	
	Швеция	031-987 65 43	
	Бразилия	(11) 555 4640	
	Россия	(095) 998-4510	
	Германия	(069) 992755	Plutzer (на Web)

Рис. 10.21. Редактирование гиперссылки

Далее редактированием можно изменить содержимое текста гиперссылки. Как видно из рис. 10.21, описание гиперссылки находится в начале строки и отделено от основного адреса символом «#».

10.6. Использование языка SQL

Для построения и выполнения произвольной запросной функции в Access очень удобным и доступным является язык запросов по образцу QBE, поддержанный мощным интерфейсом пользователя. Язык SQL в некотором роде скрыт от пользователя, хотя весьма важен при составлении VBA-программ в приложениях Access. В этом подразделе опишем, как используется язык SQL в различных объектах БД Access. Краткая справочная информация по реализованному в Access диалекту SQL приводится в Приложении 2.

Особенности применения запросов SQL

Напомним, что **запросом SQL** называют запрос, создаваемый с помощью инструкции SQL. Примерами запросов SQL являются запросы на объединение, запросы к серверу, управляющие и подчиненные запросы.

Запрос на объединение – это такой запрос, в котором объединяются поля (столбцы) одной или нескольких таблиц или запросов в одно поле или столбец в результирующем наборе записей. Например, шесть продавцов каждый месяц представляют руководству описи имеющихся товаров. Создав запрос на объединение, можно объединить эти описи в результирующем наборе записей, а затем разработать запрос на создание таблицы, основанный на запросе на объединение.

Запрос к серверу выполняет передачу через ODBC команд SQL-серверу, например, Microsoft SQL Server. Запросы к серверу позволяют непосредственно работать с таблицами на сервере вместо их присоединения. Результатом выполнения запроса к серверу может быть загрузка записей или изменение данных.

Управляющий запрос создает или изменяет объекты базы данных, такие как таблицы Access или SQL Server.

Подчиненный запрос состоит из инструкции SQL SELECT, находящейся внутри другого запроса на выборку или запроса на изменение. Эти инструкции вводятся в строку «Поле» бланка запроса для определения нового поля или в строку «Условие отбора» для определения условия отбора поля. Подчиненные запросы используются для выполнения следующих действий:

- проверка в подчиненном запросе существования некоторых результатов с помощью зарезервированных слов EXISTS или NOT EXISTS;
- поиск в главном запросе любых значений, которые равны, больше или меньше значений, возвращаемых в подчиненном запросе (с помощью зарезервированных слов ANY, IN или ALL);
- создание подчиненных запросов внутри подчиненных запросов (вложенных подчиненных запросов).

Язык SQL в Access может применяться при разработке экранных форм, отчетов, а также при создании макрокоманд и программ на VBA.

Связь языков QBE и SQL

В Access между языками QBE и SQL имеется тесная связь. Запросные таблицы (бланки, формы) на языке QBE, заполняемые пользователем, перед непосредственным выполнением преобразуются в выражения (или сообщения) SQL. То есть язык SQL является внутренним стандартом на выполнение запросов. Такой механизм имеет преимущество, поскольку позволяет внутри системы Access унифицировать подготовку запросов к выполнению на локальном и удаленном компьютерах. В последнем случае SQL-сообщение реально передается к компьютеру-серверу запроса.

В Access запрос может находиться в одном из трех режимов (состояний): Конструктора, SQL и таблицы. Режим Конструктора применяют для разработки нового запроса с чистого листа (без использования Мастеров или других средств) или для изменения макета существующего запроса. Режим SQL применяют для ввода или просмотра инструкций SQL. Режим таблицы применяют для работы с результатами выполнения запроса.

В режим таблицы запрос переходит при выборе нужного запроса во вкладке Запросы (Queries) окна БД и нажатии кнопки Открыть (Open). Нажатие кнопки Конструктор (Design) или Создать (New) в окне БД переводит запрос в режим Конструктора. В режим SQL можно попасть из других режимов по команде основного меню Вид | Режим SQL (View | SQL).

Приведем пример состояний для одного и того же запроса. Предположим, мы открыли БД «Борей», в которой нас интересует запрос под названием «Десять самых дорогих товаров» (рис. 10.22).

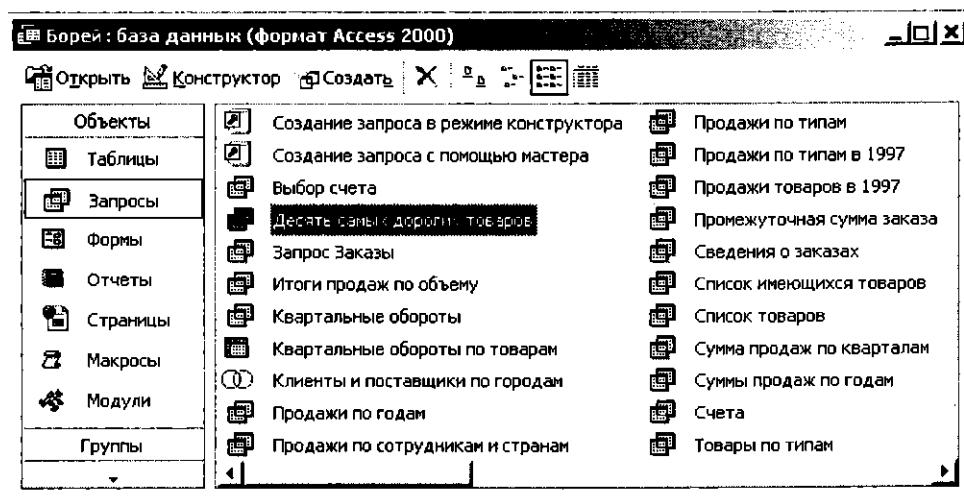


Рис. 10.22. Окно примера БД «Борей»

После открытия запроса можно увидеть результат его выполнения (рис. 10.23).

Самые Дорогие Товары		Цена
►	Dote de Blaye	1 185 750,00р.
	Thuringer Rostbratwurst	567 055,00р.
	Mishi Kobe Niku	436 500,00р.
	Sir Rodney's Marmalade	364 500,00р.
	Carnarvon Tigers	281 250,00р.
	Raclette Courdavault	247 500,00р.
	Manjimup Dried Apples	238 500,00р.
	Tarte au sucre	221 850,00р.
	Ipoх Coffee	207 000,00р.
*	Rossle Sauerkraut	205 200,00р.

Запись: [◀] [◀] [▶] [▶] [▶] [▶] Всего: 10

Рис. 10.23. Результат выполнения запроса «Десять самых дорогих товаров»

Перейдя в режим Конструктора, видим структуру запроса в виде бланка QBE (рис. 10.24).

Поле:	СамыеДорогиеТовары	Цена	
Имя таблицы:	Товары	Товары	
Сортировка:	по убыванию		
Вывод на экран:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Условие отбора:	<input type="text"/>		
или:	<input type="text"/>		

Рис. 10.24. Запрос на языке QBE

В режиме SQL в окне отображается формулировка запроса в виде инструкций на языке SQL (рис. 10.25).

```
■ Десять самых дорогих товаров : запрос на выборку
SELECT DISTINCTROW TOP 10 Товары.Марка AS
СамыеДорогиеТовары, Товары.Цена
FROM Товары
ORDER BY Товары.Цена DESC;
```

Рис. 10.25. Запрос на языке SQL

SQL в формах и отчетах

Основными источниками записей в экранных формах и отчетах являются таблицы и запросы. Во втором случае запросом может быть готовый запрос к БД или создаваемый при разработке формы или отчета. Описать новый запрос как источник записей при разработке формы или отчета в режиме Конструктора можно следующим образом.

1. Открыть окно базы данных и на вкладке **Формы (Forms)** или **Отчеты (Reports)** нажать кнопку **Создать (New)**.
2. В очередном окне выбрать режим создания объекта с помощью Конструктора и нажать **OK**.
3. В окне создания новой формы или отчета подвести курсор мыши к заголовку окна и в контекстном меню выбрать пункт **Свойства (Properties)**.
4. В появившемся окне **Форма (Form)** или **Отчет (Report)** на вкладке **Данные (Data)** определить источник записей с помощью SQL-выражения, вводимого в поле ввода свойства «Источник записей», или с помощью Построителя запросов, вызываемого нажатием кнопки напротив поля ввода SQL-выражения. При вызове Построителя запросов появляется окно добавления таблиц и запросов в модель запроса (рис. 10.26).
5. Ввести нужные таблицы/запросы в схему запроса-источника записей, получив тем самым формируемый запрос, возможный вид которого показан на рис. 10.27.

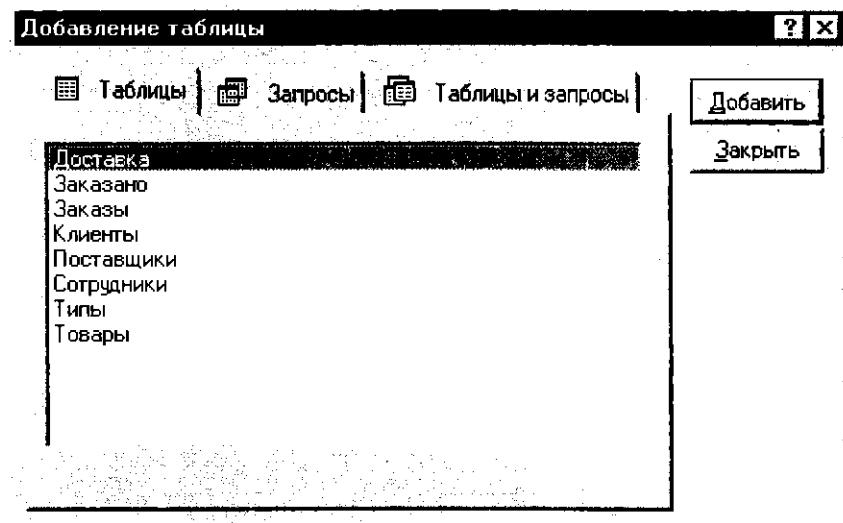


Рис. 10.26. Окно добавления таблиц и/или запросов

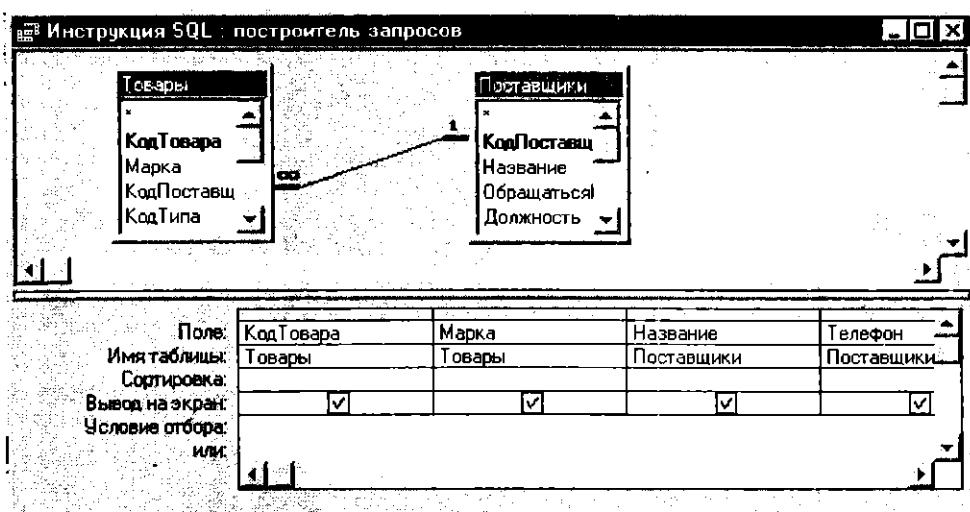


Рис. 10.27. Вид формируемого запроса

От этого запроса легко перейти к SQL-выражению одним из описанных способов, например, по команде Вид | Режим SQL (View | SQL).

SQL в макрокомандах

Макрокоманды входят в состав макросов, которые используются для автоматизации выполнения часто повторяющихся действий в работе с БД. Макрос представляет собой одну или несколько макрокоманд с аргументами.

Макросы вызываются из окна БД или автоматически при наступлении определенных событий. В последнем случае требуется связать макрос с нужным событием. Событием, по которому вызывается макрос, может быть, например, нажатие кнопки в области экранной формы или открытие окна БД.

Наряду с выполнением некоторых действий над объектами БД макросы могут вызывать другие макросы, программы на Visual Basic и внешние приложения.

Из множества макрокоманд (около 50) с SQL непосредственно связаны две макрокоманды: ЗапускЗапросаSQL (RunSQL) и ОткрытьЗапрос (OpenQuery).

Макрокоманда ЗапускЗапросаSQL запускает запрос на изменение или управляющий запрос Access с помощью соответствующей инструкции SQL. Эта макрокоманда делает возможным выполнение действий в макросе без предварительного создания сохраненных запросов. С помощью макрокоманды можно выполнять и сохраненные запросы.

Запросами на изменение являются инструкции SQL (подраздел 3.9), реализующие следующие функции: добавление (**INSERT INTO**), удаление (**DELETE**), создание таблицы (**SELECT...INTO**) и обновление (**UPDATE**).

Управляющими запросами являются инструкции SQL, выполняющие следующие функции: создание таблицы (**CREATE TABLE**), изменение таблицы (**ALTER TABLE**), удаление таблицы (**DROP TABLE**), создание индекса (**CREATE INDEX**) и удаление индекса (**DROP INDEX**).

Единственным и обязательным аргументом макрокоманды *ЗапускЗапросаSQL* является инструкция SQL. Максимальная длина инструкции SQL составляет 255 символов. Для выполнения инструкции SQL длиной более 255 символов следует вызвать метод RunSQL объекта DoCmd в программе VBA. В программах VBA допускается использование инструкций SQL длиной до 32768 символов.

Аргумент макрокоманды в виде текста SQL-инструкции вводится вручную в окне ввода макрокоманды или копируется из окна SQL, что часто удобнее. Для выполнения последней манипуляции можно поступить так: войти в режим Конструктора запросов, создать запрос, получить эквивалентный оператор SQL, выделить его и поместить в буфер обмена. Пример готовой к выполнению макрокоманды с оператором SQL приведен на рис. 10.28.

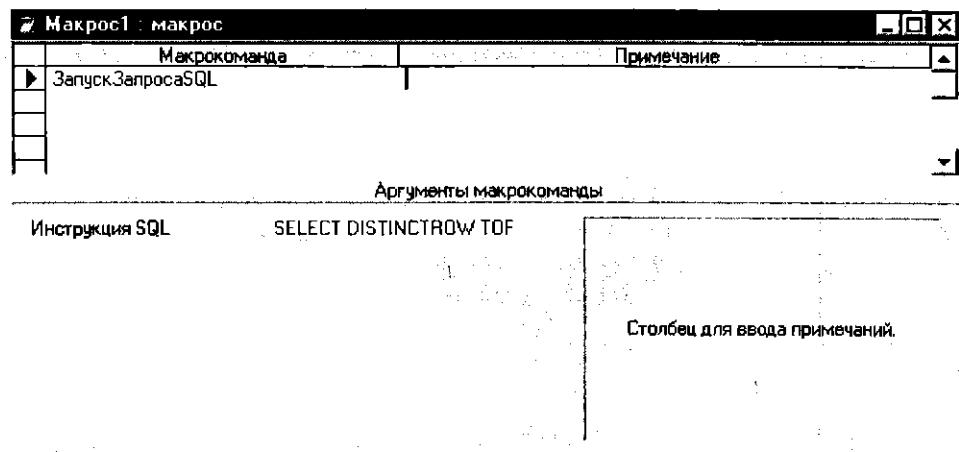


Рис. 10.28. Макрокоманда с оператором SQL

Макрокоманда *ОткрытьЗапрос* позволяет открывать запрос на выборку или перекрестный запрос (в режимах Таблицы, Конструктора и Предварительного просмотра), запускать запрос на изменение или ввод данных. Вызов ее эквивалентен нажатию кнопки **Открыть** (Open) или Конструктор (Design) в окне базы данных после выбора запроса на вкладке **Запросы** (Queries).

В макрокоманде задаются три аргумента: имя запроса, режим и режим данных. Первый аргумент задает имя открываемого запроса и является обязательным. Второй аргумент задает режим открытия запроса. Допустимыми значениями являются: «Таблица» (используется по умолчанию), «Конструктор» и «Просмотр».

Третий аргумент описывает режим ввода данных в запрос. Аргумент можно применять к запросам, открываемым в режиме таблицы. Допустимые значения: «Добавление» (можно вводить новые записи, нельзя изменять существующие), «Изменение» (можно вводить новые и изменять существующие записи; действует по умолчанию) и «Только чтение» (разрешается просматривать записи).

Для ускоренного создания макроса с этой макрокомандой можно воспользоваться следующим приемом. Выбрать запрос в окне базы данных и переместить с помощью мыши в строку макрокоманды в макросе. При этом в макрос автоматически добавляется макрокоманда *ОткрытьЗапрос*, открывающая запрос в режиме таблицы.

Для вызова макрокоманды *ОткрытьЗапрос* в программе VBA используют метод OpenQuery объекта DoCmd.

SQL в программах на VBA

Напомним, что программы на VBA, как и макросы, предназначены для автоматизации выполнения повторяющихся операций над объектами БД Access. Для выполнения программы следует запустить содержащую ее процедуру **Sub** или функцию **Function**. Программа записывается в процедуре как набор инструкций и методов, с помощью которых выполняются требуемые действия. Процедуры, как известно, хранятся в модулях и выполняются в ответ на события или вызываются из выражений, макросов и других процедур.

В Access существуют следующие способы запуска программ VBA:

- включение программы в процедуру обработки события;
- вызов функции в выражении;
- вызов процедуры **Sub** в другой процедуре или в окне отладки;
- выполнение макрокоманды ЗапускПрограммы (**RunCode**) в макросе.

Например, чтобы вызвать программу выполнения определенных действий при открытии формы, нужно включить ее текст в процедуру обработки события Нажатие кнопки (**Click**) для кнопки, при нажатии которой будет открываться форма.

Функции применяются в выражениях, определяющих вычисляемые поля в формах, отчетах или запросах. Выражения используются для указания условий в запросах и фильтрах, а также в макросах, в инструкциях и методах VBA, а также в инструкциях SQL. В процедуру **Sub** можно включать общедоступные VBA-подпрограммы, вызываемые из других процедур.

Рассмотрим **выполнение запроса** к базе данных с помощью инструкций SQL в программе на Visual Basic для приложений.

В запросе производится отбор в базе данных записей, удовлетворяющих определенным условиям (запрос на выборку), либо выдается инструкция на выполнение указанных действий с записями, удовлетворяющими определенным условиям (запрос на изменение).

Если запрос SQL используется для возвращения данных, ядром базы данных Microsoft Jet создается объект **Recordset**. После создания этого объекта можно использовать методы поиска (**Find**) и перемещения по записям набора.

Существуют следующие способы выполнения запросов:

- вызов метода **Execute** (для выполнения запросов SQL на изменение);
- создание и выполнение специального объекта **QueryDef**;
- использование инструкции SQL в качестве аргумента метода **OpenRecordset**;
- выполнение метода **OpenRecordset** для существующего объекта **QueryDef**;
- вызов методов **RunSQL** и **OpenQuery**.

Метод Execute используется, если требуется выполнить такое изменение в БД, при котором не возвращаются записи. Это, например, операции вставки или удаления записей.

В качестве простейшего примера приведем команды Visual Basic для приложений выполнения запроса на изменение, в котором выполняется обновление записей таблицы «Должности», не имеющих значение в столбце ISBN. При возникновении ошибки все изменения отменяются.

```
Dim strSQL as String
strSQL = «DELETE FROM Должности WHERE ISBN IS NULL»
dbsBiblio.Execute strSQL, dbFailOnError
```

Объект QueryDef представляет собой сохраненное определение запроса в базе данных. Его можно рассматривать как откомпилированную инструкцию SQL.

Приведенная ниже программа выполняет создание нового объекта QueryDef, после чего соответствующий запрос открывается в режиме таблицы.

```
Sub NewQuery()
    'Объявление переменных
    Dim dbs As Database, qdf As QueryDef, strSQL As String
    'Установка значения переменной типа Database,
    'представляющей текущую базу данных
    Set dbs = CurrentDb
    'Строка запроса
    strSQL = «SELECT * FROM Сотрудники WHERE
    [ДатаНайма] >= # 1-1-95#»
    'Создание нового объекта QueryDef
    Set qdf = dbs.CreateQueryDef(«НовыеСотрудники», strSQL)
    'Открытие запроса
    DoCmd.OpenQuery qdf.Name
End Sub
```

Метод OpenRecordSet используется, чтобы открыть объект типа RecordSet для выполнения последующих операций над ним.

В следующей процедуре с помощью инструкции SQL создается объект Recordset типа динамического набора записей. В предложение WHERE инструкции SQL включена функция Year, определяющая отбор заказов, размещенных в 1998 году.

```
Sub Orders98()
    Dim dbs As Database, rst As Recordset, strSQL As String
```

```

Dim fld As Field
Set dbs = CurrentDb
strSQL = «SELECT DISTINCTROW Заказ, ДатаРазмещения « &
    «FROM Заказы WHERE ((Year([ДатаРазмещения])=1998));»
Set rst = dbs.OpenRecordset(strSQL, dbOpenDynaset)
rst.MoveLast
Debug.Print rst.RecordCount
End Sub

```

Метод RunSQL выполняет макрокоманду ЗапускЗапросаSQL (RunSQL) в программе VBA. В следующем примере изменяется название должности всех агентов по продажам в таблице «Сотрудники».

```

DoCmd.RunSQL «UPDATE Сотрудники « &
    «SET Сотрудники.Title = 'Региональный представитель' « &
    «WHERE Сотрудники.Title = 'Агент по продажам';»

```

Метод OpenQuery выполняет макрокоманду ОткрытьЗапрос (OpenQuery) в программе VBA. С его помощью можно открыть запрос в режиме таблицы, Конструктора или просмотра. При этом устанавливается один из следующих режимов работы с данными: добавление, изменение или только чтение.

В следующем примере запрос «Выработка сотрудников» открывается в режиме таблицы, в котором пользователю разрешается просмотр записей.

```
DoCmd.OpenQuery «Выработка сотрудников», , acReadOnly
```

Выбор варианта выполнения запросов определяется программистом с учетом особенностей решаемой задачи.

10.7. Защита баз данных

В Access реализованы следующие способы защиты баз данных: парольная защита, защита на уровне пользователей и шифрование.

Парольная защита БД

Парольная защита является простым и часто достаточным средством обеспечения защиты БД от открытия несанкционированными пользователями. Используемый при этом пароль называют *паролем базы данных*.

Зная пароль БД, любой пользователь сможет ее открыть и использовать, а также выполнить все необходимые операции с ней. Установка пароля может

быть запрещена в случае, если для БД установлена защита на уровне пользователя и наложен запрет на парольную защиту.

Парольная защита может использоваться в дополнение к защите на уровне пользователя. В этом случае устанавливать парольную защиту может пользователь, обладающий правами администратора БД.

Поскольку парольную защиту можно применять наряду с защитой на уровне пользователя, возникает вопрос: «Можно ли защитить БД паролем, а затем защитить ее на уровне пользователя?». Ответ — нет. Дело в том, что защита на уровне пользователя состоит в создании новой БД, которая защищена и имеет структуру исходной БД. Из-за того, что Access не позволяет защищенному на уровне пользователя БД именовать так же, как и исходную, то новая БД всегда будет без пароля. После открытия БД ее можно легко защитить с помощью пароля. Таким образом, для установки обоих видов защиты БД сначала защищают на уровне пользователя, а потом устанавливают пароль.

Если парольная защита действует наряду с защитой на уровне пользователя, то пользователю предоставляется возможность выполнять над объектами БД действия, предусмотренные правами доступа.

Парольную защиту БД нельзя использовать в случае, если предполагается выполнять репликацию БД. Система Access не позволяет создавать копии (реплики) БД, защищенных паролем.

Метод парольной защиты достаточно надежен, так как пароль система Access шифрует, и к паролю нет прямого доступа. Он хранится вместе с защищаемой БД, а поэтому открыть БД не удается и на другом компьютере с установленной там системой.

Увидеть пароль или найти место в файле БД, которое он занимает, — занятие неперспективное. Попытки авторов сделать это с помощью средств прямого редактирования файлов (например, утилитой *diskedit* комплекта утилит *Norton Utilities*) и специально для этого разработанных программ на языке Паскаль ни к чему не привели. При использовании парольной защиты от пользователя требуется подобрать удачный пароль и надежно его сохранить от потери и от хищения.

Процедура установки парольной защиты БД включает следующие шаги.

1. Закрытие базы данных, если она открыта. Если база данных используется в сети, следует проверить, что все остальные пользователи тоже закрыли ее.
2. Выбор в меню команды **Файл | Открыть (File | Open)**. Появится диалоговое окно **Открытие файла базы данных (Open)**.
3. Установка с помощью соответствующего списка в правой нижней части окна открытия файла режима монопольного доступа (**Монопольно (Exclusive)**) и открытие базы данных.

4. Выбор команды Сервис | Защита | Задать пароль базы данных (Tools | Security | Set Database Password).
5. Ввод пароля в поле Пароль (Password) с учетом регистра клавиатуры.
6. Подтверждение введенного пароля путем повторного его ввода в поле Подтверждение (Verify), а после этого — нажатие OK.

Удалить пароль намного проще, главное — знать его при открытии БД. Для удаления пароля БД следует выполнить четыре действия.

1. Открыть базу данных в режиме монопольного доступа (см. установку пароля выше).
2. Из меню системы выдать команду Сервис | Защита | Удалить пароль базы данных (Tools | Security | Unset Database Password). Команда доступна, если пароль базы данных уже установлен.
3. В поле Пароль (Password) появившегося окна Удалить пароль базы данных ввести текущий пароль.
4. Нажать OK. База данных по-прежнему остается открытой. При очередном ее открытии система Access запрашивать пароль не будет.

Средств изменения пароля БД в Access нет, поэтому для изменения пароля следует удалить пароль, а затем определить новый. При использовании парольной защиты нужно учитывать, что она не защищает БД от удаления.

Если пароль утерян, то доступ к защищенной БД не получить. Попавшую в такое положение БД остается удалить. Для удаления БД можно воспользоваться, например, программой MS Windows Проводник (Windows Explorer). Удалить БД, не выходя из Access, можно так: вызвать окно открытия файла базы данных, выделить удаляемую БД, вызвать контекстное меню и выбрать пункт Удалить (Delete). После выполнения операции удаления БД последняя попадет в Корзину.

Защищая паролем, следует иметь в виду, что в случае *связывания таблиц* БД, защищенных паролем, могут быть проблемы. Пусть таблица БД (назовем ее подключаемой БД) связывается с таблицей защищенной паролем БД. При образовании связи система Access потребует от пользователя ввода пароля защищенной БД. Введенный пароль системой в некотором месте запоминается. Если в последующем пароль защищенной БД изменится, то доступ к защищенным таблицам подключаемой БД будет невозможен. Система Access информирует об изменении пароля.

Защита на уровне пользователя

Защита на уровне пользователя применяется в случаях, когда с одной БД работают несколько пользователей или групп пользователей, имеющих разные права доступа к объектам БД. Использовать защиту на уровне пользова-

теля можно на отдельном компьютере и при коллективной работе в составе локальной сети. Этот способ защиты подобен способам разграничения доступа в локальных сетях.

Для организации защиты на уровне пользователя в системе Access создаются рабочие группы (РГ). Каждая рабочая группа определяет единую технологию работы совокупности пользователей. Система Access в произвольный момент времени может работать с одной РГ. Заметим, что СУБД может работать с одной базой данных. Если в сеансе работы СУБД в ней попытаться открыть вторую БД, то первая БД автоматически закроется. Правда, можно запустить на выполнение несколько систем Access, поскольку Windows является многооконной системой.

Информация о каждой РГ хранится в соответствующем файле РГ (System.mdw), который автоматически создается при установке системы. Информация о размещении этого файла хранится в системном реестре. Для управления рабочими группами в Access 2002 имеется программа «Администратор рабочих групп» (ARG), запустить которую можно из подменю **Сервис | Защита (Tools | Security)**.

Кроме сведений о системе защиты на уровне пользователя в файле РГ хранятся параметры системы Access. Изменить установленные по умолчанию системные параметры пользователь может с помощью команды **Сервис | Параметры (Tools | Parameters)**. Эти параметры включают в себя: параметры отображения информации системой Access (строки состояния, окна запуска, панели инструментов, скрытых и системных объектов), параметры средств разработки запросов, экранных форм, отчетов и программ (модулей), установку действий на нажатие клавиш, параметры режима правки и поиска информации в БД и другие параметры, определяющие режим открытия БД по умолчанию (общий или монопольный доступ), вид блокировки при совместном изменении информации в БД (отсутствие, всех записей или только изменяемой записи), период обновления и т. д.

Файл РГ описывает группы пользователей и отдельных пользователей, входящих в эту РГ. Он содержит учетные записи групп пользователей и отдельных пользователей. По каждой учетной записи система Access хранит права доступа к объектам базы данных.

По умолчанию в каждую рабочую группу входит две группы пользователей: администраторы (имя группы **Admins**) и обычные пользователи (имя группы **Users**). Причем, в группу **Admins** первоначально включен один администратор под именем **Admin**.

При создании групп указывается имя (идентификатор) группы и код, представляющий собой последовательность от 4 до 20 символов. При регистрации (создании) пользователей в системе защиты им присваивается имя, код и необязательный пароль. Имена групп и пользователей, их коды, а также пароли пользователей (если они заданы) Access скрывает от пользователе-

ля. Поэтому если пользователь их забудет, найти их в файле РГ и восстановить практически невозможно. Коды группы и пользователя используются для шифрования системой учетных записей в файле РГ.

Каждому из пользователей, независимо от принадлежности к группе, можно присвоить пароль. Этот пароль, в отличие от пароля БД, называется **паролем учетной записи** и хранится в учетной записи в файле РГ. Первоначально все включаемые в РГ пользователи, в том числе и пользователь Admin, имеют пустой пароль или, можно считать, что не имеют пароля.

Каждой из групп приписываются определенные права на объекты БД. Члены группы Admins имеют максимальные права.

Наличие двух функциональных групп пользователей в рабочей группе, как правило, достаточно для организации нормальной работы коллектива пользователей. При необходимости можно создать дополнительные группы пользователей. Один пользователь может входить в состав нескольких групп. При подключении пользователя, зарегистрированного в нескольких группах, действуют минимальные из установленных в разных группах ограничения на объекты БД.

При создании рабочих групп и регистрации пользователей действуют ограничения, к основным из которых относятся следующие.

1. Группы Admins и Users удалить невозможно.
2. В группе Admins должен быть хотя бы один пользователь. Первоначально таким пользователем является пользователь Admin (администратор). Удалить пользователя Admin из этой группы можно после включения в нее еще одного пользователя.
3. Все регистрируемые пользователи автоматически становятся членами группы Users. Удалить их из этой группы нельзя.
4. Удалить пользователя Admin из рабочей группы нельзя (из группы Admins его можно удалить, а из группы Users — нет).
5. Создаваемые группы не могут быть вложены в другие группы, другими словами, нельзя создавать иерархию групп пользователей.
6. В системе защиты могут быть пустые группы, но не может быть пользователей, не входящих ни в одну группу (они обязательно войдут в группу Users).

Рабочая группа имеет структуру, показанную на рис. 10.29. Здесь буквами А, В и F обозначены созданные группы пользователей, а буквами Р1, Р2, Р3, Р4 и Рn — пользователи. Все пользователи являются членами группы Users. Группа F пока пуста.

Основное назначение системы защиты на уровне пользователя состоит в контроле прав доступа к объектам базы данных. Для этого нужно установить защиту БД с помощью Мастера защиты.

Существующие в Access типы прав доступа приведены в таблице 10.1.

Права подключающегося пользователя делятся на **явные и неявные**. Явные

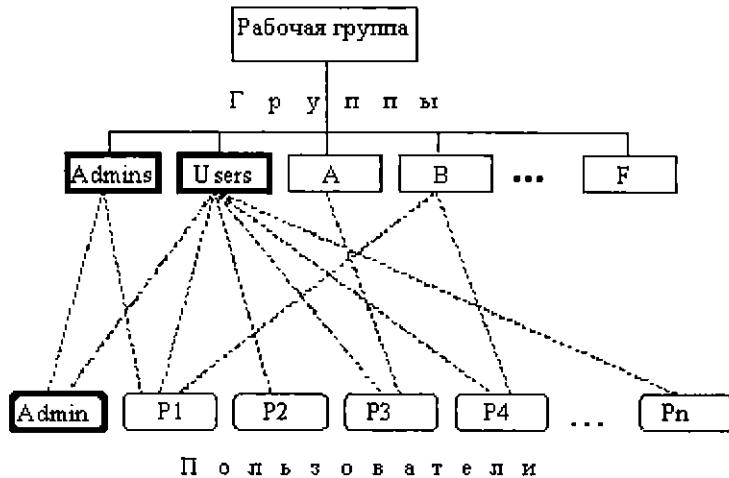


Рис. 10.29. Структура рабочей группы

Таблица 10.1.

Типы прав доступа

Права доступа	Разрешенные действия	Объекты
Открытие/запуск (Open/Run)	Открытие базы данных, формы или отчета, запуск макроса	Базы данных, формы, отчеты и макросы
Монопольный доступ (Open Exclusive)	Открытие базы данных для монопольного доступа	Базы данных
Чтение макета (Read Design)	Просмотр объектов в режиме Конструктора	Таблицы, запросы, формы, отчеты, макросы и модули
Изменение макета (Modify Design)	Просмотр и изменение макета объектов или удаление объектов	Таблицы, запросы, формы, отчеты, макросы и модули
Администратора (Administer)	Для баз данных: установка пароля, репликация и изменение параметров запуска. Для объектов базы данных: полные права на объекты и данные, в том числе предоставление прав доступа	Базы данных, таблицы, запросы, формы, отчеты, макросы и модули
Чтение данных (Read Data)	Просмотр данных	Таблицы и запросы
Обновление данных (Update Data)	Просмотр и изменение данных без их вставки и удаления	Таблицы и запросы
Вставка данных (Insert Data)	Просмотр и вставка данных без их изменения и удаления	Таблицы и запросы
Удаление данных (Delete Data)	Просмотр и удаление данных без их изменения и вставки	Таблицы и запросы

права имеются в случае, если они определены для учетной записи пользователя. Неявные права — это права той группы, в которую входит пользователь. Напомним, что пользователь, имеющий явные и неявные права в одной или нескольких группах, пользуется правами с минимальными ограничениями из всей совокупности прав.

Изменить права других пользователей на объекты некоторой БД могут следующие пользователи:

- члены группы *Admins*, определенной в файле РГ, использовавшемся при создании базы данных;
- владелец объекта;
- пользователь, получивший на объект права администратора.

Изменить свои собственные права в сторону их расширения могут пользователи, являющиеся членами группы *Admins* и владельцы объекта.

Владельцем объекта считается пользователь, создавший объект. Владельца объекта можно изменить путем передачи права владельца другому пользователю. Правами изменения владельца обладают пользователи, имеющие право изменить права доступа к объекту или создать объект.

Передать права новому владельцу можно по команде меню **Сервис | Защита | Разрешения (Tools | Security | User And Group Permissions)**.

Создать объект в новой БД, аналогичный объекту существующей базы данных, можно с помощью операций импорта/экспорта или копирования его в базу данных. Для создания и защиты на уровне пользователя копии всей БД удобно воспользоваться Мастером защиты, вызов которого выполняется командой меню **Сервис | Защита | Мастер (Tools | Security | User-Level Security Wizard)**. Владельцем защищенной БД может стать другой пользователь. В этом случае тоже говорят об изменении владельца. При этом у исходной БД остается свой владелец.

Установка защиты на уровне пользователя сложнее парольной защиты. Она предполагает создание файлов РГ, учетных записей пользователей и групп, включение пользователей в группы, активизацию процедуры подключения к Access, а также защиту каждой из БД с помощью Мастера защиты. Рассмотрим коротко эти операции.

Как отмечалось, *манипуляции с файлами РГ* выполняются с помощью **программы АРГ**. Назначение программы АРГ состоит в «привязке» Access к нужному файлу РГ. Запустить программу можно с помощью выбора команды из меню системы **Сервис | Защита (Tools | Security)**. Программа АРГ имеет три функции (и соответствующие кнопки): создание файла РГ, связь с произвольным файлом РГ и выход из программы.

Система защиты на уровне пользователя действует в двух **основных режимах**: с требованием подключения пользователя к системе; без требования подключения пользователя.

В первом случае при попытке выполнить какую-нибудь операцию (например, открыть БД) после запуска Access открывается диалоговое окно **Вход (Logon)** и от пользователя требуется ввести свое имя и пароль, если он имеется. Чтобы система Access всегда запрашивала имя и пароль, достаточно установить пароль пользователю Admin. При снятии пароля окно входа не появляется.

Если окно входа в систему неактивно (пользователь Admin не имеет пароля), то при запуске Access автоматически подключает пользователя с именем Admin и предоставляет тому, кто запустил систему, имеющиеся у пользователя Admin права. Этот вариант защиты менее эффективен, так как подключиться к системе может любой пользователь. Единственное, что можно сделать – это ограничить его права по доступу к объектам БД.

Отметим, что защита в системе устанавливается относительно имеющихся в Access баз данных. Подключиться к системе несложно. Иногда для этого надо знать пароль пользователя Admin. Если он пароля не имеет, то и этого не нужно.

Для активизации окна входа в систему нужно выполнить следующее.

1. Запустить Access.
2. Командой Сервис | Защита | Пользователи и группы (Tools | Security | User and Group Accounts) открыть окно Пользователи и группы (рис. 10.30).

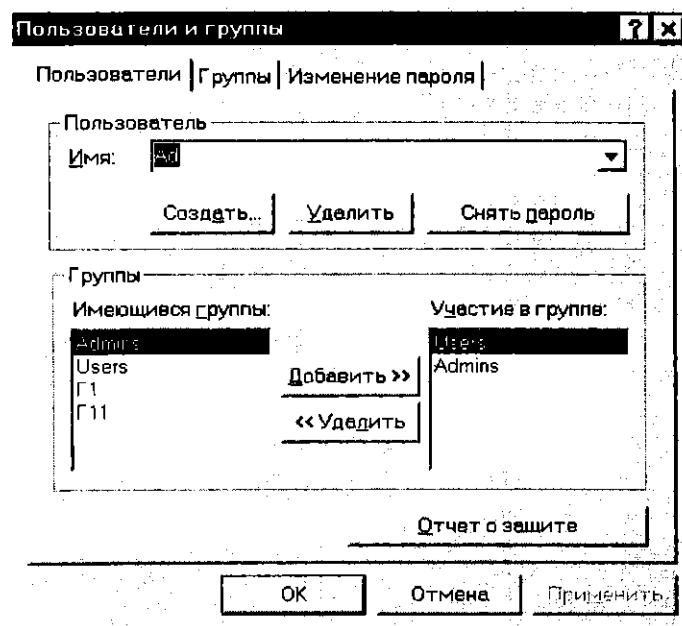


Рис. 10.30. Окно Пользователи и группы

3. На вкладке Пользователи (Users) проверить, что в поле со списком Имя (Name) выбрана стандартная учетная запись пользователя «Admin».
4. На вкладке Изменение пароля (Change Logon Password) оставить пустым поле Текущий пароль (Old Password), а в поле Новый пароль (New Password) ввести новый пароль. Вводимые символы отображаются звездочками (*). Пароли могут содержать от 1 до 14 любых символов с учетом регистра, кроме символа с кодом ASCII 0 (пустой).
5. Подтвердить пароль, повторно введя его в поле Подтверждение (Verify), и нажать OK.

Чтобы отключить окно входа в систему описанным способом, нужно изменить текущий пароль пользователя Admin на пустой. Для этого достаточно на вкладке Пользователи (Users) в поле Имя (Name) выбрать строку Admin и нажать кнопку Снять пароль (Clear Password).

Замечания.

- Для того чтобы обойти требование ввода пароля администратора в Access, достаточно указать системе на использование другого файла рабочей группы, в котором оно не установлено. В простейшем случае — это создаваемый при установке системы файл system.mdw. Чтобы система использовала его, а не текущий файл, достаточно запустить программу администратора рабочих групп и с его помощью связать Access с другим файлом. Файл рабочей группы не обязательно должен быть «родным». Несовпадение имени и названия организации не мешает нормальному запуску системы с другим файлом рабочей группы, который может находиться в любой папке.
- Чтобы обезопасить себя от порчи файла рабочей группы или утери пароля администратора из группы Admins, следует сохранить исходный файл рабочей группы на диске. В критический момент следует подключиться к нужному файлу.
- При организации работы нескольких групп пользователей на одном компьютере целесообразно для каждой из групп создать свой файл рабочей группы. Перед началом работы следует подключиться к нужному файлу. Чтобы случайно не подключиться к чужому файлу и не дать возможности несанкционированному пользователю войти в систему Access, файл лучше хранить отдельно от системы.

Перед **созданием** учетных записей пользователей, составляющих группы, если встроенных групп Admins и Users недостаточно, нужно создать учетные записи самих групп.

Создаваемые учетные записи пользователей автоматически попадают в группу пользователей Users. Если необходимо, чтобы пользователи принад-

лежали другим группам, в частности к группе пользователей-администраторов Admins, следует включить в эти группы учетные записи пользователей.

При изменении конфигурации рабочей группы приходится удалять или вставлять соответствующие учетные записи. Эти операции выполняются в диалоговом окне, показанном на рис. 10.30.

Для работы с группами выбирается вкладка **Группы (Groups)**. Функция **создания** новых групп доступна для пользователей, подключившихся к Access при запуске как членов группы Admins. В диалоговом окне нужно ввести имя группы и ее код. Код не является паролем и в процессе дальнейшей работы системы Access используется при шифрации информации для группы. Самому пользователю код не нужен, и вводить его при действиях с БД не требуется. Его желательно сохранить на случай утраты файла РГ, когда потребуется вновь описывать группу пользователей. В противном случае не удастся получить доступ к защищенной БД.

Удаление группы выполняется с помощью кнопки **Удалить (Remove)**. Стандартные группы Admins и Users не удаляются.

Для работы с **учетными записями** отдельных пользователей выбирается вкладка **Пользователи (Users)**. Создание и удаление учетной записи осуществляется с помощью кнопок **Создать (New)** и **Удалить (Remove)** соответственно. При удалении учетной записи пользователя она удаляется из всех групп. Удалить учетную запись пользователя Admin невозможно.

Чтобы включить пользователя в группу, достаточно выбрать учетную запись этого пользователя в поле **Имя (Name)**, затем выбрать в списке **Имеющиеся группы (Available Groups)** нужную группу, нажать кнопку **Добавить (Add)**. Выбранная группа появляется в списке **Участие в группе (Member of)**.

Для удаления пользователя из группы нужно после выбора имени пользователя в поле **Имя (Name)** и имени группы из списка **Участие в группе (Member of)** нажать кнопку **Удалить (Remove)**. Удалить пользователя Admin из группы Users невозможно, а удалить его из группы Admins можно при наличии в этой группе хотя бы еще одного пользователя.

Возможность непосредственного изменения учетных записей пользователей и групп отсутствует, для этого следует пользоваться функциями удаления и создания.

С помощью кнопки **Отчет о защите (Print Users and Groups)** в окне **Пользователи и группы (User and Group Accounts)**(рис. 10.30) можно получить отчет различной степени детальности:

- о пользователях и группах;
- о пользователях;
- о группах.

Для получения отчета надо выполнить два условия: открыть базу данных, чтобы проверить право пользователя на доступ к ней, а также включить принтер, так как отчет выводится на бумагу.

Определение прав пользователей и групп в Access выполняется в соответствии с правами подключенного к системе пользователя и открытой базой данных.

Вызов окна управления правами доступа (рис. 10.31) выполняется с помощью команды Сервис | Защита | Разрешения (Tools | Security | User And Group Permissions).

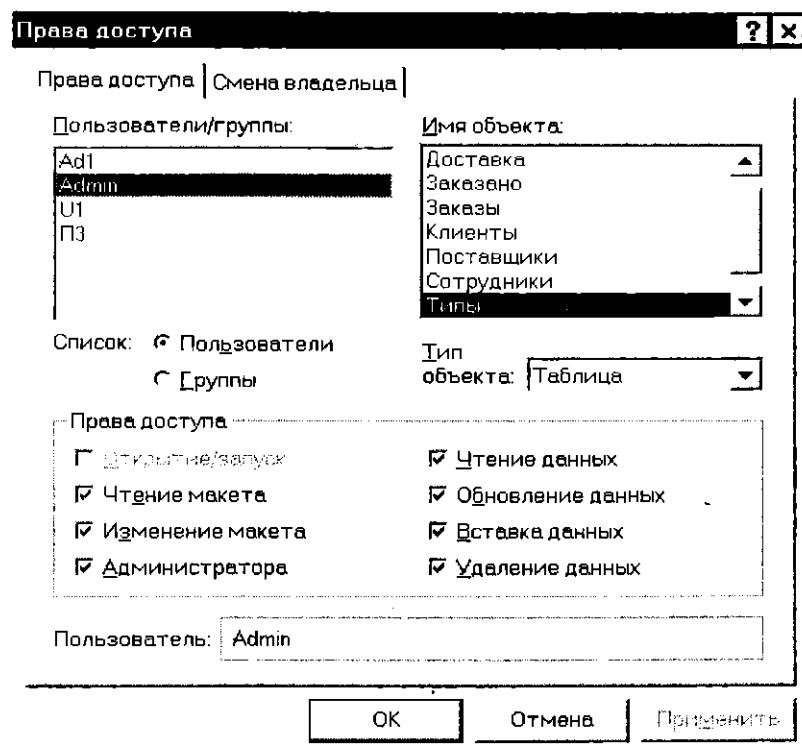


Рис. 10.31. Окно Права доступа

Окно имеет две вкладки: **Разрешения** (User And Group Permissions) и **Смена владельца** (Change Owner). Установка прав может выполняться по отношению к группам и отдельным пользователям. Выбор управляется с помощью переключателей в группе Список (List).

При управлении правами доступа учитывают следующее.

1. Возможности манипуляций зависят от полномочий текущего пользователя, указанного в нижней части окна.
2. Для выбора нескольких объектов в области Имя объекта (Object Name) можно провести указатель мыши при нажатой левой кнопке или выбирать объекты клавишей управления курсором при нажатой клавише <Ctrl>.

3. Кнопкой **Применить (Apply)** пользуются для того, чтобы не выходить из окна после очередной установки.
 4. Выход из окна происходит при нажатии **OK**.
- Для смены владельца некоторого объекта БД служит вкладка **Права доступа (User And Group Permissions)** (рис. 10.32).

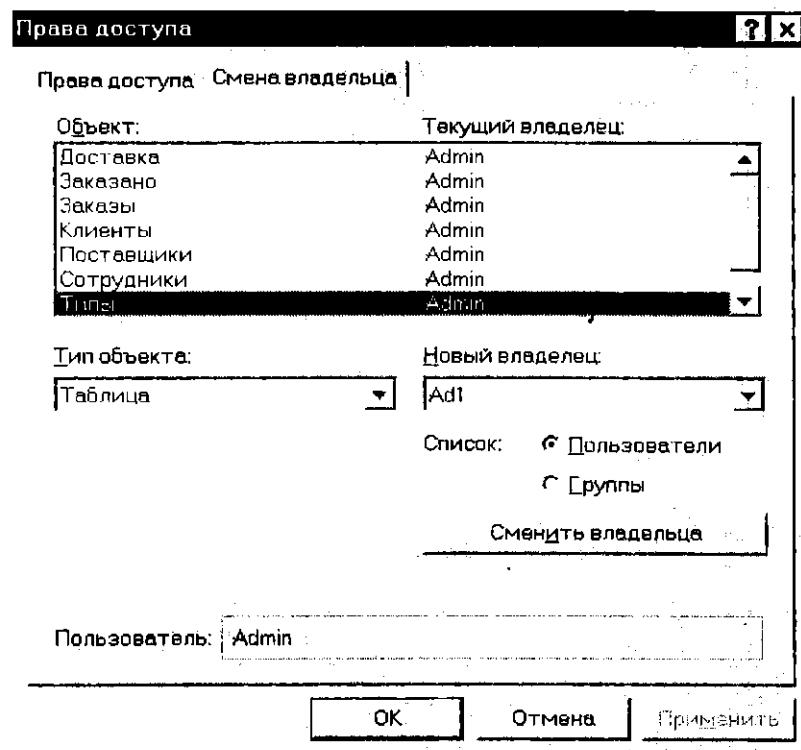


Рис. 10.32. Вкладка Права доступа

Новым владельцем может быть отдельный пользователь и группа. Это зависит от установки переключателя в области **Список (List)**. Если права владельца передаются учетной записи группы, то права владельца автоматически получают все пользователи из этой группы. Возможности операций по смене владельца зависят от полномочий текущего пользователя. Нажатие кнопки **Сменить владельца (Change Owner)** позволяет назначать различным объектам в зависимости от их типов новых владельцев.

Установка защиты базы данных на уровне пользователя выполняется с помощью Мастера защиты. Создание структуры рабочей группы желательно завершить до вызова Мастера защиты. Результат защиты — создание новой

защищенной БД, для объектов которой можно изменять права доступа других пользователей.

Замечание.

Мастер защиты является необходимым средством установки защиты базы данных на уровне пользователя. Образовать защищенную на уровне пользователя БД другими способами не удается.

Применяться Мастер защиты может любым пользователем, открывшим незащищенную БД, кроме пользователя Admin (даже если лишить его прав администрирования базы данных, которую предполагается защитить на уровне пользователя). Если попытаться защитить базу данных, то получить можно обычную копию исходной базы. После образования новой защищенной БД, правами администратора обладает создавший ее пользователь — он же и вызывал Мастера. Члены группы Admins такими правами после создания БД не обладают, но могут легко их присвоить.

При вызове Мастера защиты указывается тип защищаемых объектов БД. Защищаемые от пользователей объекты указываются при установке прав доступа.

Для вызова Мастера защиты следует открыть БД и выполнить команду Сервис | Защита | Мастер (Tools | Security | User-Level Security Wizard). Появится диалоговое окно, показанное на рис. 10.33. В этом окне нужно оп-

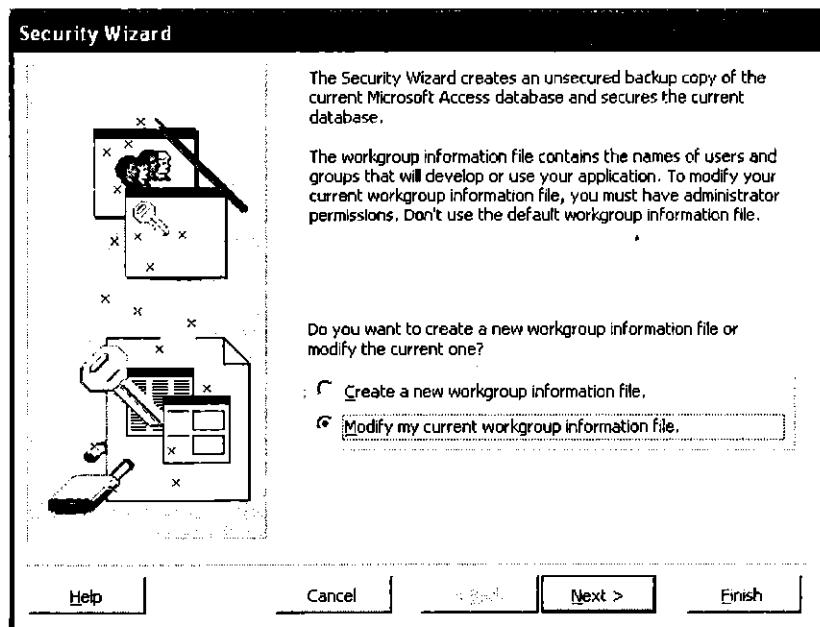


Рис. 10.33. Диалоговое окно Мастера защиты

ределить, будет ли создаваться новый файл РГ или мастер должен скорректировать существующий и использующийся в настоящее время файл. В первом случае после выполнения защиты следует не забыть о связывании Access с вновь созданным файлом РГ.

В последующих окнах мастер предоставит возможность указать полное имя файла РГ, выбрать защищаемые объекты, определить стандартные группы, включаемые в файл РГ, назначить права рабочим группам, создать учетные записи пользователей и включить их в группы, а также определить имя копии незащищенной базы данных.

Таким образом, новая защищенная БД будет иметь имя исходной БД со всеми объектами, в том числе и межтабличными связями. Для предоставления другим пользователям прав доступа к защищенным объектам следует открыть защищенную БД и выполнить команду Сервис | Защита | Разрешения (Tools | Security | User And Group Permissions).

Снять защиту на уровне пользователя для БД можно в текущей рабочей группе или во всех рабочих группах. В первом случае, для снятия защиты достаточно подключиться к Access с правами администратора (как любой пользователь группы Admins) и передать все права на объекты БД группе Users. Поскольку все пользователи группы являются членами этой группы, то они получают полный доступ к БД.

Полное снятие защиты, при котором доступ к БД имеют пользователи всех групп, состоит в создании новой БД с полными правами для группы Users. Это можно реализовать, если сначала выполнить операции по снятию защиты в своей рабочей группе, а затем подключиться к Access как администратор Admin, создать пустую БД, в которую импортировать объекты исходной БД. После этого следует удалить исходную базу данных, в противном случае в системе будет две одинаковые БД: защищенная и незащищенная.

Процедура снятия защиты для БД в текущей рабочей группе предполагает выполнение следующих действий.

1. Запуск Access и подключение к ней от имени администратора (члена группы Admins).
2. Открытие базы данных.
3. Предоставление группе «Users» полных прав на все объекты в базе данных.
4. Выход из системы.

Процедура снятия защиты для БД во всех рабочих группах включает перечисленные действия, а также следующее:

1. Подключение к системе с именем «Admin».
2. Создание пустой базы данных.

3. Импортирование в новую открытую базу данных всех объектов из исходной БД с помощью команды **Файл | Внешние данные | Импорт (File | Get External Data | Import)**.
4. Удаление исходной БД после проверки правильности снятия защиты.

Для *удаления БД* нужно выполнить команду **Файл | Открыть (File | Open)**. В появившемся диалоговом окне найти БД, выделить ее, вызвать контекстное меню и задать команду **Удалить (Delete)**. Для удаления БД можно также воспользоваться стандартной программой MS Windows **Проводник (Windows Explorer)**.

Для организации защиты рекомендуется следующее:

- спланировать работу пользователей, объединив их в одну или несколько рабочих групп;
- в каждой рабочей группе выделить группы пользователей;
- каждого пользователя из каждой рабочей группы зарегистрировать в системе защиты, присвоив имя и необязательный пароль;
- активизировать окно входа в систему Access, иначе вход в нее будет проходить от имени администратора Admin. Если он не будет иметь пароля, то можно считать, что защиты нет, так как любой пользователь может войти в систему с большими полномочиями;
- защитить требуемые базы данных с помощью Мастера защиты. Запускает Мастера защиты обычно тот, кто создал БД или предполагающий стать ее владельцем;
- удалить каждую исходную БД, позаботившись о создании ее копии или копии защищенной БД;
- каждой группе (не рекомендуется, но можно, и каждому пользователю) при необходимости присвоить полномочия по доступу к объектам каждой защищенной БД.

Для организации доступа нескольких групп пользователей к объектам одной защищенной БД следует поступить следующим образом.

С помощью Мастера защиты защитить базу данных. Затем в одной из РГ создать группу (несколько групп, если предполагаются различные права пользователей), имя и код группы (групп) запомнить. После этого в каждой РГ создать учетную запись группы (групп) с такими же именем и кодом группы (групп) защищенной БД. Идентичность учетных записей в разных РГ дает возможность доступа различных РГ к одной общей БД. Доступ пользователей различных групп к одной и той же БД становится возможным после включения их в созданную группу (группы). В каждой РГ при необходимости следует установить права доступа к объектам БД. В последующем состав групп и права в группах можно менять.

Замечания.

- Система защиты Access позволяет создавать произвольное число групп пользователей, но создавать большое число групп не стоит. Возможно, что достаточно двух стандартных групп: **Admins** и **Users**.
- Права доступа пользователей лучше назначать на уровне групп. Это упрощает использование системы защиты для администраторов и пользователей.
- Для усиления защиты БД, кроме защиты на уровне пользователя, можно также защитить ее паролем. Пароль БД задается после установки защиты на уровне пользователя с помощью Мастера защиты. Для получения доступа при открытии базы данных пользователями вводится пароль, а при работе с объектами БД они имеют права доступа группы с учетом личных прав, если они определены.
- При организации защиты следует беречь файлы РГ от копирования потенциальными злоумышленниками. Файл для каждой из рабочих групп рекомендуется хранить на отдельном носителе (дискете).

Шифрование баз данных

Средства шифрования в Access позволяют кодировать файл БД таким образом, что она становится недоступной для чтения из других программ, в которых известен формат БД Access.

Шифровать незащищенную паролем базу данных большого смысла нет, так как дешифровать БД может любой пользователь этой или другой ПЭВМ, где установлена система Access. Более того, пользователь может открыть и использовать зашифрованную БД, как и обычную незашифрованную.

По-видимому, шифрация в Access применяется, чтобы изменить до неузнаваемости стандартный формат файла БД. При выполнении процедуры шифрации/дешифрации БД не нужно задавать ключ шифрации (он формируется или зашифтуется в системе Access, изменяется ли он в зависимости от версии, лицензионного номера пакета и других параметров — авторам неизвестно). Невозможность устанавливать ключ, по нашему мнению, сводит на нет всю мощь механизма шифрации в Access.

Для шифрации/дешифрации базы данных требуется выполнить следующее.

1. Запустить Access. Для выполнения операций надо обладать правами владельца базы данных. Нельзя зашифровать БД, открытую и используемую в сети другими пользователями.
2. Выдать команду Сервис | Защита | Шифровать/десифровать (Tools | Security | Encrypt/Decrypt Database).
3. Указать имя базы данных, которую требуется зашифровать или десифровать, и нажать OK.

4. Указать имя, диск и папку для целевой (зашифрованной) базы данных и нажать OK.

При низком качестве магнитных дисков (чаще всего дискет) надо быть осторожным во время шифрации БД. Возможно, что зашифрованной БД присваивается имя, совпадающее с исходной БД (исходная БД «затирается» зашифрованной), а после этого выясняется, что при дешифрации из-за искажения информации возникли ошибки, не дающие доступа к базе данных.

Во избежание этого рекомендуется шифруемой БД задавать имя, отличное от исходного имени. Затем проверить правильность операции шифрации путем пробной работы с зашифрованной БД. Если все нормально, исходную БД можно удалить, а зашифрованную БД – переименовать в исходную.

Если попытаться зашифровать БД, защищенную паролем, то для этого необходимо в ответ на запрос Access ввести соответствующий пароль.

Для обычной работы с зашифрованной БД ее не обязательно специально расшифровывать. Система «понимает» и зашифрованную информацию. Следует иметь в виду, что с зашифрованной базой данных Access работает несколько медленнее, поскольку операции шифрации/десифрации выполняются в реальном масштабе времени.

10.8. Скрытие объектов баз данных

Механизм скрытия объектов применяется в случаях, когда пользователь работает с базой данных через стандартный интерфейс — окно БД, и желательно предохранить базу данных от случайного доступа к ее объектам.

Скрываемые от пользователя объекты не удаляются, а становятся временно невидимыми. Скрывать от пользователя можно произвольные объекты различных типов: таблицы, формы, запросы, отчеты, макросы и модули.

Для скрытия текущего объекта БД надо в окне свойств этого объекта установить флажок атрибутов Скрытый (Hidden). Окно свойств можно вызвать с помощью кнопки Свойства (Properties) или команды контекстного меню. В любой момент времени текущим может быть один объект, поэтому скрыть несколько объектов одновременно нельзя, это делается последовательно.

Еще один способ сделать произвольный объект БД скрытым — это переименовать его таким образом, чтобы его имя начиналось с символов «Usys» (на любом регистре: Usys, usys, USYS и т. д.). Переименование текущего объекта выполнить просто: достаточно вызвать меню его свойств и выбрать пункт Переименовать (Rename).

Скрытые объекты БД пользователь видит в окне базы, если в текущих установках параметров Access не задано отображение скрытых объектов в окне БД.

Чтобы скрытые объекты сделать не скрытыми, нужно сначала сделать их видимыми. Для этого достаточно в основном меню системы из пункта Сервис (Tools) выбрать пункт Параметры (Parameters) и на вкладке Вид (View) появившегося окна в области Отображение на экране (View) установить флажок Скрытые объекты (Hidden Objects). Все скрытые объекты появляются в окне БД. Пиктограммы скрытых объектов, расположенные слева от их имен, отличаются от не скрытых объектов более светлым цветом. После этого нужно вызвать окно свойств каждого из объектов и сбросить флажок атрибутов Скрытый (Hidden).

Кроме пользовательских объектов, в каждой БД есть системные или служебные объекты. Имена этих объектов начинаются с символов «MSys». Например, среди служебных таблиц имеются следующие: MSysQueries, MSysRelationships, и другие. Атрибут этих объектов — «системный». Обычно они не отображаются в окне БД, хотя не относятся к скрытым. Чтобы их увидеть, нужно в основном меню системы из пункта Сервис (Tools) выбрать пункт Параметры (Parameters) и на вкладке Вид (View) появившегося окна в области Отображение на экране (View) установить флажок Системные объекты (System Objects).

10.9. Обслуживание баз данных

Основными способами обслуживания БД в Access 2002 являются следующие: копирование, восстановление и сжатие (компрессия) баз данных.

Копирование баз данных применяется для защиты их от случайной потери. Для создания копий можно использовать также репликацию.

Создавать копии БД можно различными средствами: стандартной программой MS Windows Проводник (Windows Explorer), программой Norton Commander и другими подобными программами. Отметим, что копирование защищенной паролем БД в папку Портфель (My Briefcase) с помощью программы невозможно, так как в этом случае делается попытка создать реплику БД. Как отмечалось ранее, это недопустимо. Программа Проводник (Explorer) «знает» об особенностях использования защищенных БД в Access.

Копии БД можно хранить в сжатом виде, для чего используют программы-архиваторы. Это позволяет сэкономить дисковое пространство, но увеличивает время на получение архивной копии и восстановление информации из архивов.

Если размеры файлов БД велики, можно применять специальные программы разбиения файлов на части. Многие архиваторы наряду со сжатием информации позволяют создавать многофайловые архивы.

Восстановление применяется при повреждениях БД, не позволяющих пользователю нормально работать с базой данных или даже открыть ее. Од-

ной из причин повреждения может быть воздействие компьютерных вирусов или наличие дефектов (физических или логических) на диске. Весьма вероятной причиной повреждения файла базы данных является выключение питания компьютера до предварительного закрытия сеанса работы с базой данных Access.

Повреждение базы данных Access в большинстве случаев определяется при попытках пользователя открыть, сжать, зашифровать или дешифровать БД. В некоторых ситуациях сразу не удается определить, что база данных повреждена. Если база данных ведет себя непредсказуемо, то, скорее всего, она требует своего восстановления.

Сжатие базы данных средствами Access отличается от сжатия с помощью архиваторов и состоит в освобождении места на диске от удаленных из базы данных записей. Более подходящими, на наш взгляд, здесь являются термины «реорганизация», «реструктурирование» или «реструктуризация» БД.

Необходимость такого сжатия базы данных средствами Access обусловлена следующим. При внесении пользователем изменений в базу данных (объектов и записей в таблицах) файл БД только увеличивается. Занимаемая удаленными объектами и записями таблиц память не освобождается, а отмечается как неиспользуемая. При очередном добавлении объектов и записей снова выделяется память под эти объекты и размер файла базы данных увеличивается.

Чтобы БД не была перегружена неиспользуемыми областями («пустотой»), периодически следует ее сжимать. Эксперименты показывают, что серия последовательных созданий и удалений таблиц приводит к существенному увеличению размеров БД.

В отличие от предыдущих версий Access, где операции восстановления и сжатия выполнялись отдельно, в Access 2002 эти операции объединены. Для компрессии и восстановления данных необходимо сделать следующее:

1. Закрыть базу данных (это желательно, но вовсе не обязательно). При работе в сети убедиться, что другие пользователи тоже закрыли ее.
2. Создать резервную копию базы данных.
3. Выдать команду Сервис | Служебные программы | Сжать и Восстановить (Tools | Database Utilities | Compact and Repair Database).
4. В диалоговом окне (рис. 10.34) выбрать нужную БД и нажать кнопку Сжать (Compact).

Для сжатия или восстановления БД, защищенной паролем, естественно, требуется предварительно ввести пароль. Часто выполнять сжатие мало используемых БД не имеет смысла.

Результат сжатия можно поместить в ту же базу данных (это и есть фактическая операция сжатия), либо указать другое имя. В первом случае требуется подтвердить замену существующей базы данных. После получения под-

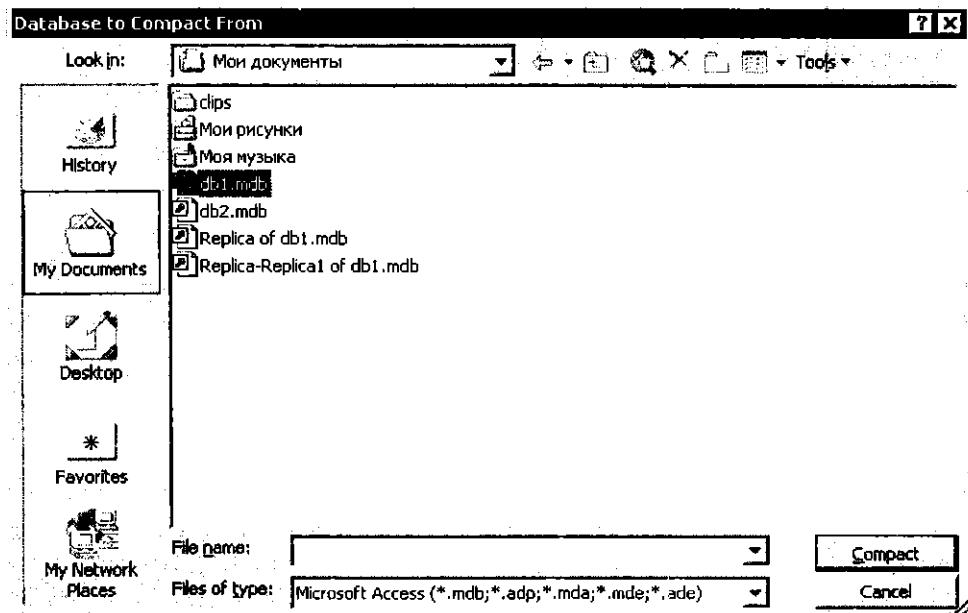


Рис. 10.34. Диалоговое окно сжатия базы данных

тверждения Access выполняет сжатие исходной базы данных во временном файле. При успешном завершении сжатия исходная база данных удаляется, а ее имя присваивается полученной сжатой копии. Во втором случае получают улучшенную копию исходной БД.

В полном смысле сжатие информации базы данных, или, более точно, сжатие файла хранения БД, выполняется с помощью архиваторов. Например, с помощью архиватора арj демонстрационную базу данных «Борей» можно сжать примерно до 30% от исходного объема.

10.10. Репликация баз данных

Репликация баз данных применяется для создания специальных «горячих» копий БД средствами Access. С помощью репликации можно легко получать новые копии БД, используемые как на одном компьютере, так и в сети. Отдельные копии (реплики) требуется периодически синхронизировать.

Понятие о репликации

Репликацией называют создание специальных копий (реплик) базы данных Access, с которыми пользователи могут одновременно работать на разных рабочих станциях. Отличие репликации от обычного копирования файлов

лов ОС (в том числе файлов БД) заключается в том, что для каждой реплики возможна синхронизация с остальными репликами. В случаях, когда для хранения реплик БД используется папка Портфель (*My Briefcase*), репликацию БД называют *портфельной репликацией*. Для использования этой репликации на рабочем столе системы должен быть создан портфель, если его там нет. Это можно сделать с помощью контекстного меню, вызванного на рабочем столе.

По одной базе данных можно создать набор реплик. В наборе различают основную реплику и дополнительные реплики. *Основная* реплика отличается от обычной (дополнительной) реплики тем, что в ней *можно изменять структуру* БД. Основную реплику можно сделать обычной, а дополнительную реплику — основной, но в любой момент времени в наборе реплик одна реплика является основной, а остальные — дополнительными. Дополнительные реплики можно создавать из основной и дополнительных реплик.

Создание основной реплики состоит в преобразовании файла исходной БД в новый файл. Исходную базу данных будем называть *реплицируемой*. Если основную реплику назвать тем же именем, что и исходная (реплицируемая) БД, то последняя пропадет. Для безопасности перед проведением преобразования целесообразно создать резервную копию исходного файла БД. Если пользователь не создал резервную копию исходной БД до начала репликации, он может это сделать в процессе ее выполнения.

В ходе репликации в файл исходной БД добавляются специальные таблицы, поля и свойства. После репликации исходная база данных становится основной репликой в наборе реплик. Основная и дополнительная реплики могут содержать реплицируемые и не реплицируемые (локальные) объекты. Полученная основная реплика с точки зрения работы пользователя не отличается от исходной БД.

При репликации базы данных Access добавляет системные таблицы (например, MsySidetables, MsySchemaProb, MSysReplicas), изменять содержимое большинства из которых пользователю не рекомендуется. Эти таблицы могут быть видимыми или невидимыми, в зависимости от того, как установлен флажок **Системные объекты** (*System Objects*) на вкладке **Вид** (*View*).

В каждую таблицу реплицированной БД добавляются следующие системные поля:

- **s_GUID** — глобальный уникальный идентификатор каждой записи;
- **s_ColLineage, s_Lineage** — двоичные поля, содержащие информацию об истории изменения записей;
- **s_Generation** — поле, содержащее информацию о групповых изменениях.

Реплицироваться могут все объекты БД: таблицы, формы, запросы, отчеты, макросы и модули. В наборе реплик реплицируемыми должны быть одни и те же объекты. Каждая из реплик может содержать свои собственные ло-

кальные объекты, структура и содержание которых не передается в другие реплики. Признак реплицируемости объекта БД устанавливается Access путем изменения свойств объектов или программно с помощью программных интерфейсов к специальным объектам (см. далее). Изменение структуры реплицируемых объектов, а также содержимого БД основной реплики по специальным командам синхронизации передается во все дополнительные реплики. Схема образования основной реплики показана на рис. 10.35.

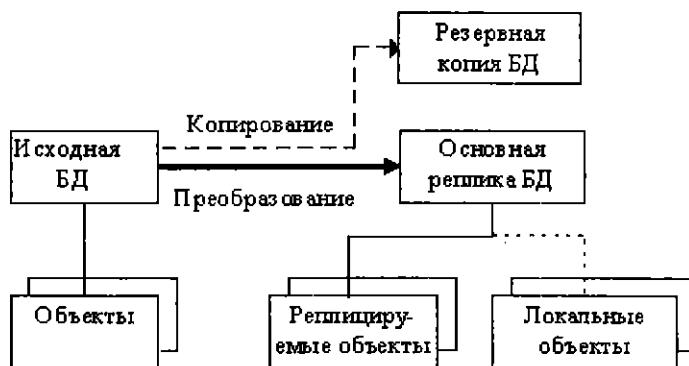


Рис. 10.35. Схема образования основной реплики

Репликация базы данных позволяет обмениваться изменениями отдельных ее копий и может применяться для следующих целей.

1. Распространение приложений. Все изменения существующих объектов и добавление новых объектов БД выполняются в основной реплике. В сеансе синхронизации между репликами последние изменения распространяются на объекты дополнительных реплик набора.
2. Доступ к данным. Пользователи, работающие на переносных компьютерах и имеющие реплики основной БД, после подключения к сети могут провести синхронизацию внесенных ими на переносных компьютерах исправлений с изменениями общей реплики.
3. Резервное копирование. Поместив реплику на другом компьютере, можно организовать резервное копирование данных основной базы. В отличие от стандартных методов копирования, которые не позволяют работать с БД во время резервного копирования, репликация разрешает вносить изменения даже во время синхронизации.
4. Перераспределение нагрузки и распараллеливание работы пользователей. Размещение реплик БД на дополнительных сетевых серверах и присоединение к ним части пользователей позволяет равномерно распределить нагрузку на серверы. Одновременная работа группы пользователей с репликами одной БД позволяет распараллелить работу и ускорить ре-

шение некоторых задач при работе с ней, например, ввод большого числа исходных данных.

Синхронизацией называют процесс обновления двух компонентов в наборе реплик, при котором производится обмен обновленными записями и объектами из каждого компонента. Access одной командой позволяет выполнить синхронизацию между двумя репликами одного набора. При этом можно синхронизировать между собой две дополнительные реплики. Схема синхронизации реплик показана на рис. 10.36.

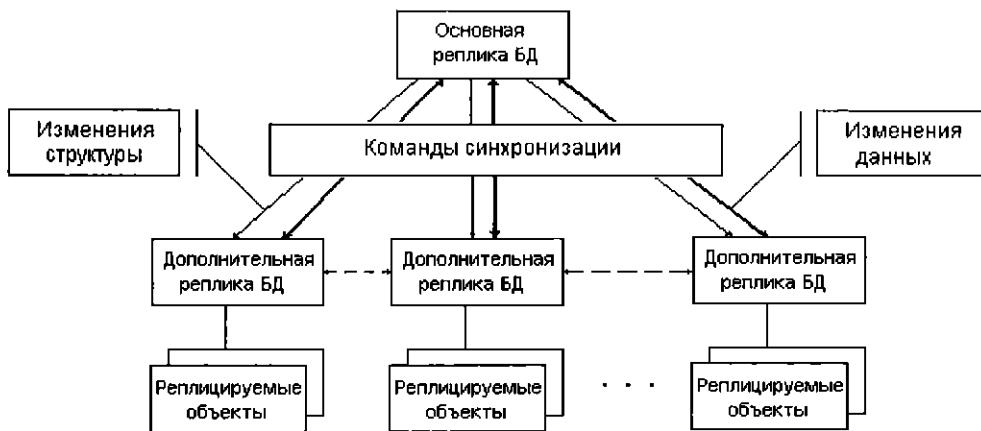


Рис. 10.36. Синхронизация реплик

После завершения синхронизации изменения, внесенные в один компонент, оказываются внесенными в другой компонент. Обе реплики становятся идентичными. Исключение могут составлять локальные объекты. В случаях, когда создается более двух реплик, для упрощения управления поддержанием соответствия структур и содержания реплик, синхронизации предпочтительно выполнять через основную реплику. В противном случае реплики набора быстро «разойдутся» и обеспечить их идентичность будет сложно.

Разумной схемой синхронизации нескольких реплик является выполнение синхронизации данных с основной репликой, избегая синхронизации напрямую с другими дополнительными репликами. При этом все последние изменения БД всегда имеются в основной реплике. Архивирование базы данных лучше свести к архивированию БД основной реплики, а не создавать множество различающихся архивов дополнительных реплик.

Синхронизацию структур набора реплик также целесообразно выполнять через основную реплику. Другими словами, управлять набором реплицируе-

мых объектов (включать и исключать их из числа реплицируемых) и их структурой (изменять реплицируемые объекты) нужно из основной реплики. Во всех репликах, в том числе основной, кроме того, допускается создание и изменение локальных объектов.

Для защиты реплицированной БД можно использовать защиту на уровне пользователя, которая позволяет определить права пользователей на ее объекты. Такие права не мешают синхронизации базы данных. Защита реплицированной базы данных, точнее, любой из ее реплик, с помощью пароля БД не допускается. Базу данных, защищенную паролем, нельзя реплицировать.

Создание и удаление реплик

Репликацию базы данных Access можно выполнять следующим образом:

- по команде Сервис | Репликация (Tools | Replication) меню Access;
- с помощью программы Проводник;
- путем вызова Диспетчера репликации пакета Microsoft Office XP для разработчиков;
- используя специальные объекты. Это могут быть объекты JRO (Jet and Replication Objects) – для работы с базами данных Access, либо объекты доступа к данным DAO (Data Access Objects) – для работы с базами MS Access версии 97 и более ранних.

В первом случае перед созданием основной реплики рекомендуется создать резервную копию исходной БД, так как последняя при репликации будет преобразована. Для создания реплики нужно открыть исходную БД и выдать команду Сервис | Репликация | Создать дополнительную реплику (Tools | Replication | Create Replica).

Система Access выдаст предупреждающее сообщение о закрытии БД. В очередном окне предлагается выбрать вариант дальнейших действий по репликации БД: создавать (Да (Yes)) или не создавать (Нет (No)) резервную копию исходной базы данных, отменить репликацию (Отмена (Cancel)) или вызвать справку (Справка (Help)). Если копия уже создана, следует нажать кнопку Нет (No). Автоматически создаваемая резервная копия исходной БД, если пользователь не изменит ее имя, хранится в той же папке, где и основная БД. Имя файла резервной копии, если пользователь его не изменит, совпадает с именем файла исходной БД, а имя файла имеет расширение bak.

В очередном окне остается определить имя и местоположение основной реплики.

Для создания дополнительной реплики достаточно открыть основную или дополнительную реплику и выдать команду Сервис | Репликация | Создать дополнительную реплику (Tools | Replication | Create Replica). В открывшемся окне указывается имя и местоположение дополнительной реплики. Таким местом может быть и Портфель (My Briefcase).

Во втором случае репликация БД выполняется из приложения Проводник (Explorer). Вызов процедуры репликации происходит при перетаскивании мышью значка исходной БД из папки в Портфель (My Briefcase). Дальнейшие действия пользователя от описанных действий сильно не отличаются.

В отличие от предыдущего случая, здесь одна из реплик остается в исходной папке, а другая – в папке Портфель (My Briefcase). Пользователю предлагается сделать выбор какую из реплик сделать основной, а какую дополнительной. Для этого в последнем предлагаемом пользователю окне, называемом Портфель (My Briefcase), необходимо отметить один из переключателей: «исходная копия» (основной репликой считать реплику в исходной папке) либо «копия в портфель» (основной репликой считать реплику в папке Портфель (My Briefcase)).

При установленном пакете Microsoft Office XP для разработчиков диспетчер реплик позволяет выполнять следующее: управлять большим набором реплик; поддерживать пользователей портативных компьютеров, которые не всегда подсоединены к сети; создавать реплики нескольких баз данных; устанавливать расписания синхронизации компонентов набора реплик; устранять ошибки и др.

Диспетчер реплик обеспечивает визуальный интерфейс для преобразования баз данных, создания реплик, просмотра связей между компонентами набора реплик, а также для установления свойств реплик.

Синхронизация реплик

Синхронизацию реплик можно выполнять при работе в Access или в среде Windows.

В первом случае нужно при открытой БД выполнить команду меню Сервис | Репликация | Синхронизация (Tools | Replication | Synchronize Now). В появившемся окне синхронизации можно определить место нахождения реплики, с которой требуется синхронизировать открытую БД, а также изменить статус одной из реплик набора. Если текущая реплика обладает правами основной, то она может легко передать свой статус той реплике, с которой осуществляется синхронизация. Последние, в свою очередь, могут легко передать статус основной другим репликам. Для изменения статуса синхронизируемых реплик достаточно отметить мышью флагок напротив предложенного системой варианта изменения статуса.

В Windows синхронизация выполняется с помощью системной папки Портфель (My Briefcase). Открытие окна портфеля происходит двойным нажатием левой кнопки мыши на соответствующем ярлыке рабочего стола.

Окно Портфель (My Briefcase) имеет строку меню, а в основной части содержит список файлов в одноименной папке. Для синхронизации отдельных файлов БД их следует отметить мышью с помощью клавиши <Ctrl> и выдать команду Портфель | Обновить (My Briefcase | Update). Находящи-

еся в папке Портфель (My Briefcase) файлы БД, как отмечалось, могут быть основными и дополнительными репликами. Появится окно, подобное приведенному на рис. 10.37.

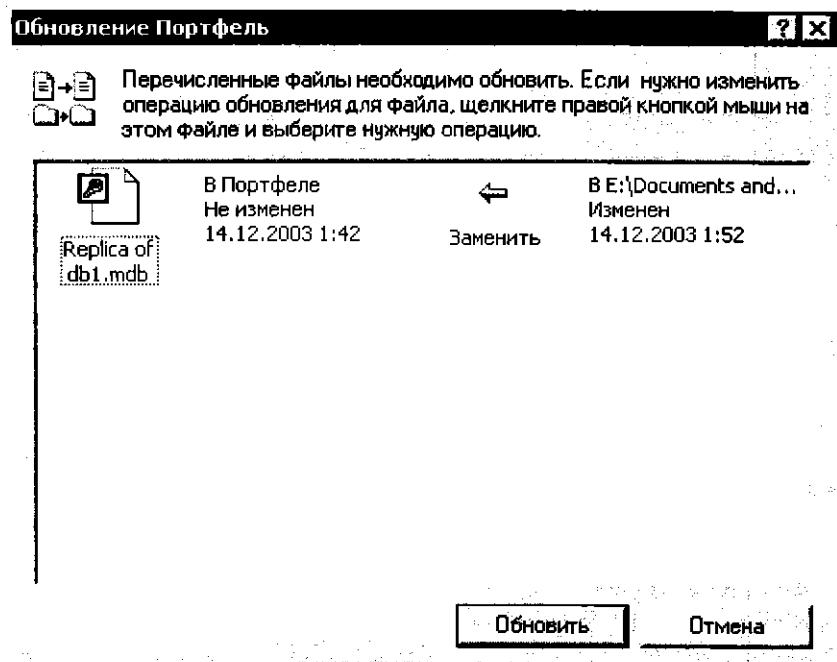


Рис. 10.37. Окно Обновление Портфель

В зависимости от изменений в каждой паре реплик система предлагает одну из операций синхронизации, основными из которых являются: замена (перенос изменений из одной изменившейся реплики в другую, не подвергшуюся изменениям), объединение (взаимное согласование изменений в репликах), создание (создание реплики в случае, если в папке Портфель (My Briefcase) оказалась обычная БД), пропуск (в списке реплик окна не обрабатывать текущую реплику). При желании с помощью мыши легко можно изменить предложенную операцию обработки любой из реплик.

Чтобы обновить реплики, хранящиеся в папке Портфель (My Briefcase), нужно выполнить команду в папке Портфель | Обновить все (My Briefcase | Update all). Появится окно, аналогичное рассмотренному ранее.

Чтобы разорвать связь одной реплики из папки Портфель (My Briefcase) с другой, нужно выделить в окне Портфель (My Briefcase) эту реплику БД и выдать команду Портфель | Отделить от оригинала (My Briefcase | Split From Original). После этого команды обновления для этой реплики становят-

ся недоступными. Синхронизация для этой реплики из Access возможна, но реальные изменения над данными этой реплики не производятся.

При независимой работе пользователей с репликами могут возникать конфликты. Чаще всего они связаны с нарушением целостности данных (например, появление одинаковых значений в ключевом поле разных реплик) или с противоречием условиям, контролирующими значения данных в таблице.

Если в процессе синхронизации система находит ошибки, пользователю об этом сообщается, а устраняет ошибки пользователь. Вызвать функцию контроля реплик на наличие конфликтов можно явно, задав команду меню Сервис | Репликация | Устраниить конфликты (Tools | Replication | Resolve Conflicts).

При обнаружении конфликтов Access выводит пользователю окно, содержащее информацию по устранению конфликтов. С его помощью можно выяснить причину и найти конфликтные записи в репликах, отобразить пояснение методов исправления ошибок.

Существенную роль при разрешении конфликтов в процессе синхронизации имеют приоритеты реплик, которые устанавливаются в момент создания реплик (кнопка Приоритет (Priority) в окне создания реплики). Приоритеты находятся в диапазоне 0–100, причем большему значению соответствует более высокий приоритет при разрешении конфликтов. В случае с одинаковыми значениями приоритетов преимущество отдается реплике с наименьшим значением свойства ReplicaID (оно доступно только для чтения и находится в системной таблице БД MSysReplicas). Если не задавать приоритет во время создания реплики, то по умолчанию создаваемая реплика будет иметь значение приоритета с коэффициентом 0.8 от значения приоритета исходной базы данных (или реплики).

10.11. Работа с мультимедиа-данными

Проблема создания и взаимодействия с мультимедиа-данными в Access скорее психологическая, нежели техническая. Для пользователей, имеющих опыт работы с СУБД предыдущих поколений (к числу которых относятся и авторы), когда не было мультимедиа, непривычно работать с мультимедиа-информацией. Психологический барьер и некоторая неуверенность в действиях — вот одна из основных причин того, что мультимедиа в базах данных не получило широкого распространения. Возможно, еще одной из причин является то, что использование мультимедиа-данных пока не стало осознанной необходимостью для пользователей. Не зная всех достоинств мультимедиа, пользователь, естественно, и не пытается их использовать.

Для знакомства с мультимедиа в СУБД Access воспользуемся одной из готовых демонстрационных БД — «Борей», поставляемой с системами Access разных версий (аналогичные возможности представляет также учебная база данных Northwind.mdb).

Мультимедиа-информация в базе данных «Борей»

В базе «Борей» мультимедиа-данные содержатся в двух таблицах: Сотрудники (поле Фотография) и Типы (поле Изображение). Поля Фотография и Изображение имеют тип поле объекта OLE (OLE Object). Основными операциями над значениями этих полей являются следующие операции: просмотр, модификация, создание и удаление.

1. Просмотр содержимого полей. Просмотреть содержимое полей Фотография и Изображение можно в режиме таблицы или формы. Для просмотра содержимого мультимедиа-полей в режиме таблицы достаточно выбрать в окне базы данных нужную таблицу, сделать соответствующее поле текущим (в области поля будет некоторый текст, свидетельствующий о том, что это поле объекта OLE, например, такой: Точечный рисунок BMP (Bitmap Image)) и дважды щелкнуть левой кнопкой мыши.

Для просмотра содержимого в режиме формы нужно открыть соответствующие экранные формы. По каждой из таблиц Сотрудники и Типы существуют формы с такими же именами. Открывая экранные формы, можно увидеть содержимое этих же полей. В общем случае это не так, поскольку мультимедиа-данные могут быть представлены в форме в виде значков, а также могут быть заданы некоторые свойства, ограничивающие или запрещающие вывод изображения на экран (такими свойствами являются задаваемые в режиме конструктора свойства «Вывод на экран» (Visible) и «Режим вывода» (Display When)). В базе данных «Борей» просмотреть поле Изображение формы Типы можно проще — с помощью Главной кнопочной формы. Для этого достаточно после вызова ее щелкнуть мышью на кнопке Типы представления этой формы. Если вы следуете приводимым советам, то перейдите к третьей записи формы Типы. В категории напитков выберите себе приз (чай, кофе или что-нибудь еще), который считайте наградой за изучение излагаемого здесь материала.

2. Модификация содержимого мультимедиа-полей. Для модификации изображений, хранящихся в полях Фотография и Изображение, независимо от режима (таблица, форма), достаточно выбрать нужное поле и дважды щелкнуть по нему кнопкой мыши. В результате будет вызван подходящий графический редактор (Paintbrush, Paint) для редактирования изображения.

3. Создание значений мультимедиа-полей. Создать содержимое мультимедиа-поля, как и другого типа, можно несколькими способами. Удобно это сделать следующим образом. Сделать это поле текущим, затем из меню Access выполнить команду Вставка | Объект (Insert | Object). Появляется диалоговое окно вставки объектов (рис. 10.38).

В нем предлагается выбрать тип объекта, а также способ его определения: создать объект с помощью соответствующей программы (Audio Recorder, Comic Chat Room, Microsoft Graph 5.0 и т. д.) или вставить его готовым из файла.

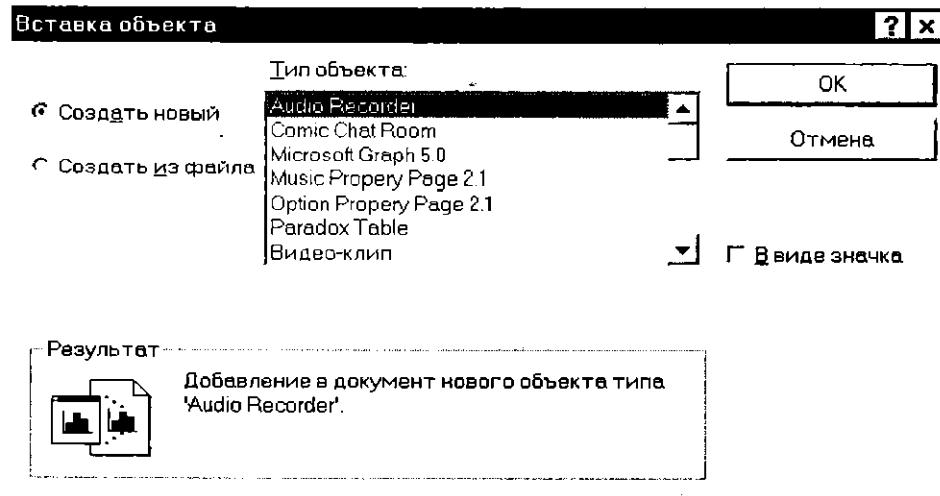


Рис. 10.38. Диалоговое окно вставки OLE-объектов

Независимо от способа определения объекта, существуют два варианта включения объекта в поле записи базы (задается с помощью флажка **Связь (Link)**):

- путем внедрения исходного объекта в базу данных;
- путем связывания (флажок **Связь (Link)** включают), когда устанавливается связь между отдельно хранящимся файлом объекта и записью базы данных.

Включаемые объекты могут отображаться при просмотре (если, конечно, они отображаются, как, например, рисунки), либо отображаться в виде значков (пиктограмм) или двойным щелчком мыши разворачиваться полностью.

4. Удаление значений мультимедиа-полей. Операция удаления значений мультимедиа-полей практически не отличается от удаления значений обычных полей. Чтобы удалить мультимедиа-поле, нужно сделать его текущим (клавишами управления курсором, клавишей <Tab>, щелчком левой кнопки мыши) и выполнить команду меню **Правка | {Вырезать, Удалить} (Edit | {Cut, Delete})**.

Работа с мультимедиа-данными в БД Access

После знакомства с готовой БД, где хранится мультимедиа-информация, сделаем некоторые обобщения.

Во-первых, для хранения мультимедиа-данных в базе необходимо в структуре таблиц БД иметь поля типа **Поле объекта OLE (OLE Object)**. Вид объекта при этом не уточняется, это значит, что в каждой записи базы данных в

этом поле может находиться свой объект: документ Word, звуковой файл, видеоклип, презентация, анимация, точечный рисунок и т. д.

Во-вторых, работать с мультимедиа-информацией можно в режиме таблицы или в режиме формы. Второй вариант предпочтителен, так как предоставляет большие возможности. Заметим, что в Access отчеты обладают практически всеми свойствами экранных форм, поэтому все, что касается форм, в равной мере относится к отчетам.

В-третьих, мультимедиа-данные хранятся в базе как встроенные (внедренные), или как связанные объекты. При этом для каждой отдельной записи базы метод включения данных может быть свой. Объект, внедренный в форму, хранится в файле базы данных. Если изменить такой объект в форме, он изменится в базе данных. Внедренный объект всегда является доступным.

При связывании изменения сохраняются в файле объекта, а не в файле базы данных. Файл объекта можно обновлять и независимо с помощью соответствующих приложений. Последние изменения выводятся на экран при следующем обращении к этим данным. Связывание удобно применять при работе с большими файлами, которые нежелательно включать в файл базы данных, а также с файлами, используемыми в нескольких формах и отчетах. При перемещении файлов связанных объектов нужно повторно устанавливать связь с ними.

В-четвертых, при разработке экранной формы требуется обратить внимание на следующие свойства мультимедиа-полей:

- **Тип вывода (Display Type)** принимает значения **Содержимое (OLE)** или **Значок (Icon)**;
- **Допустимый тип OLE (OLE Type Allowed)** принимает значения **Связанный (Linked)**, **Внедренный (Embedded)** и **Все (Either)** — допускается связывание и внедрение;
- **Доступ (Enabled)** определяет возможность модификации данных (**Да (Yes)**, **Нет (No)**).

Напомним, что определение свойств полей происходит в режиме разработки экранных форм. Простейший способ вызова окна свойств — выбор поля в области формы, нажатие правой кнопки мыши и выбор пункта **Свойства (Properties)** в контекстном меню. Перечисленные свойства находятся на вкладке **Данные (Data)** окна свойств.

Как отмечалось, в экранные формы информация может попадать из независимых источников или из таблиц БД. В справочной системе Access объекты первого вида называются свободными, а объекты второго вида — связанными (по-нашему мнению, не совсем удачно). В обоих случаях она может включаться методом внедрения или связывания. Включаемая информация на экране размещается в пределах рамки, называемой рамкой объекта.

В-пятых, в процессе работы с мультимедиа-данными допускается выполнение различных операций: создание, удаление, редактирование, а также ак-

тивизация (просмотр — для рисунков, диаграмм, документов; проигрыш — для звуковых, анимационных, видео- и других файлов).

На примере БД «Борей» мы рассмотрели, как выполняются основные операции над данными типа Точечный рисунок BMP (Bitmap Image). Более удобный и унифицированный способ выполнения операции над OLE-объектом как мультимедиа-данным, на наш взгляд, состоит в следующем:

- сделать поле текущим (таблица, форма — безразлично);
- выбрать объект щелчком левой кнопкой мыши;
- вызвать контекстное меню;
- выполнить нужную операцию: вставить новый объект, вырезать, копировать, вставить содержимое из буфера обмена или через подсекунт меню вида «Объект <тип объекта>» определить одну из таких операций, как: изменить, открыть или преобразовать объект.

10.12. Создание файлов приложений

Наряду с созданием обычных mdb-файлов приложений в Access 2002 имеется возможность создавать mde-файлы приложений, в которых хранятся БД, предназначенные «только для исполнения». Отличительной чертой этих файлов является то, что соответствующие приложения имеют ограничения по модификации их объектов, а также откомпилированы все модули и удалены исходные тексты VBA-программ. Это позволило защитить приложение от исправления и исследования, а также уменьшить размер файла БД.

Степень уменьшения размера исходной базы данных зависит, главным образом от ее состава и состояния (размеров неиспользуемых областей в файле). Так, например, размер mde-файла базы данных Northwind.mdb в формате Access 2002 меньше исходного mdb-файла на 4%.

Чтобы создать копию приложения с БД, предназначенную только для исполнения, нужно выполнить команду Сервис | Служебные программы | Создать MDE-файл (Tools | Database Utilities | Make MDE File). Появится диалоговое окно, в котором вводится имя исходной базы данных и нажимается кнопка Создать MDE (Make MDE).

В очередном диалоговом окне указывается имя новой БД и папка ее размещения. После нажатия кнопки Сохранить (Save) в окне Access откомпилирует исходную базу и сохранит ее в новом файле с расширением mde, после чего удалит из этого файла все исходные тексты VBA-программ и выполнит сжатие файла.

Если открыть mde-файл некоторой БД, то обнаружится, что в окне базы данных на вкладках Формы (Forms) и Отчеты (Reports) станут недоступными (изменят свой цвет на серый) кнопки Конструктор (Design) и Создать (New), а на вкладке Модули (Modules) — все три кнопки Запуск (Run), Конструктор (Design) и Создать (New). Это означает, что нельзя просматривать,

изменять или создавать формы, отчеты и модули, добавлять, удалять или изменять ссылки на используемые библиотеки или базы данных, изменять исполняемый код, а также импортировать или экспорттировать формы, отчеты и модули. Запустить программу модуля можно только через интерфейс приложения БД. Вместе с тем, по-прежнему остаются доступными все средства просмотра, создания и модификации таблиц, запросов и макросов.

10.13. Страницы доступа к данным

Страница доступа к данным (СДД) является специфическим видом Web-страницы, предназначенной для работы с данными, размещенными в Интернете или интранете. Используемые данные могут храниться в базах данных Access или SQL Server, а также в файлах других источников информации, например, в электронных таблицах Excel.

Характеристика страниц доступа к данным

Разработка СДД напоминает разработку форм и отчетов. В процессе разработки можно использовать список полей таблиц базы данных, окно инструментов, элементы управления, диалоговые окна сортировки и группировки и т. д. Однако имеются существенные отличия в способах разработки и непосредственной работы с данными (табл. 10.2).

Таблица 10.2.
Краткая характеристика отличий СДД от форм и отчетов

Задача	Форма	Отчет	Динамический отчет	Страница доступа к данным
Ввод, редактирование и интерактивное взаимодействие с данными в базе данных или проекте	Да	Нет	Нет	Да
Ввод, редактирование и интерактивное взаимодействие с реальными данными (<i>live data</i>) в Интернете или интранете за рамками базы данных или проекта	Нет	Нет	Нет	Да
Печать данных для распространения	Возможно	Да	Да	Возможно
Передача данных по электронной почте	Нет	Нет	Да (статические данные)	Да (реальные данные)

Примечание.

В таблице «Да» означает наилучшее выполнение задачи, «Возможно» — возможность менее оптимального решения задачи, «Нет» — невозможность решения задачи.

С учетом состава включенных элементов СДД по назначению условно можно разделить на три основных типа: получения интерактивного отчета, ввода и анализа данных соответственно.

Первый тип СДД часто используется для объединения и группирования информации в базе данных и публикации итоговых данных. Данные в СДД этого типа можно сортировать и фильтровать, но нельзя редактировать.

Второй тип СДД используется для просмотра, редактирования, добавления и удаления записей из БД. Между страницей и данными устанавливается динамическая связь с помощью операторов языка SQL, размещаемых на странице. Обмен информацией между страницей и базой данных происходит при выполнении SQL-операторов, которые обрабатываются Web-сервером, посылающим запросы к БД.

СДД третьего типа могут включать список PivotTable (сводная таблица), подобный форме PivotTable системы Access или отчету PivotTable табличного процессора Excel. Кроме того, страницы могут содержать диаграммы или электронные таблицы, в которых можно вводить и редактировать данные, а также использовать формулы как в Excel.

Страница доступа к данным сохраняется в отдельном файле вне файла базы данных.

Создание страниц доступа к данным

Начать создание СДД можно различными способами при создании базы данных и в существующей базе данных. Мы коротко рассмотрим, как создаются новые пустые СДД.

Если после запуска Access в окне справа присутствует панель задач (как на рис. 10.1), то для создания пустой СДД в разделе достаточно пойти по ссылке Пустая страница доступа к данным (Blank Data Access Page). Появлением панели управляет флажок на вкладке Вид (View) меню Сервис | Параметры (Tools | Options).

Перейти к созданию СДД можно также с помощью команды Файл | Создать (File | New) или по нажатию кнопки Создать (New) панели инструментов. В этих двух случаях появляется вышеупомянутая панель, если ее не было до этого.

Результатом перехода по ссылке Пустая страница доступа к данным (Blank Data Access Page) будет окно выбора источника данных (рис. 10.39).

При нажатии на кнопку Открыть (Open) запустится мастер соединения с данными (рис. 10.40), позволяющий установить соединение с источником данных для СДД.

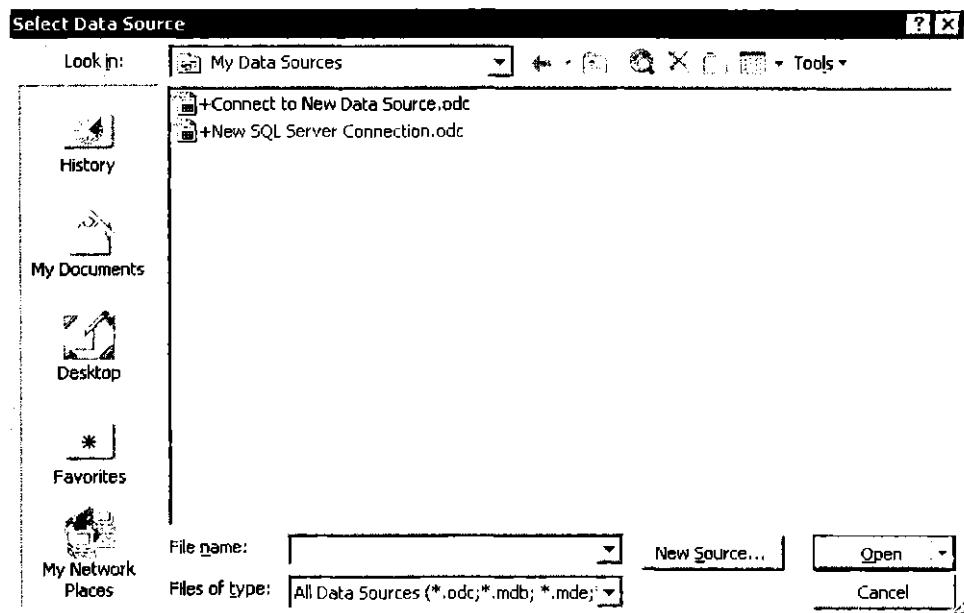


Рис. 10.39. Окно выбора источника данных

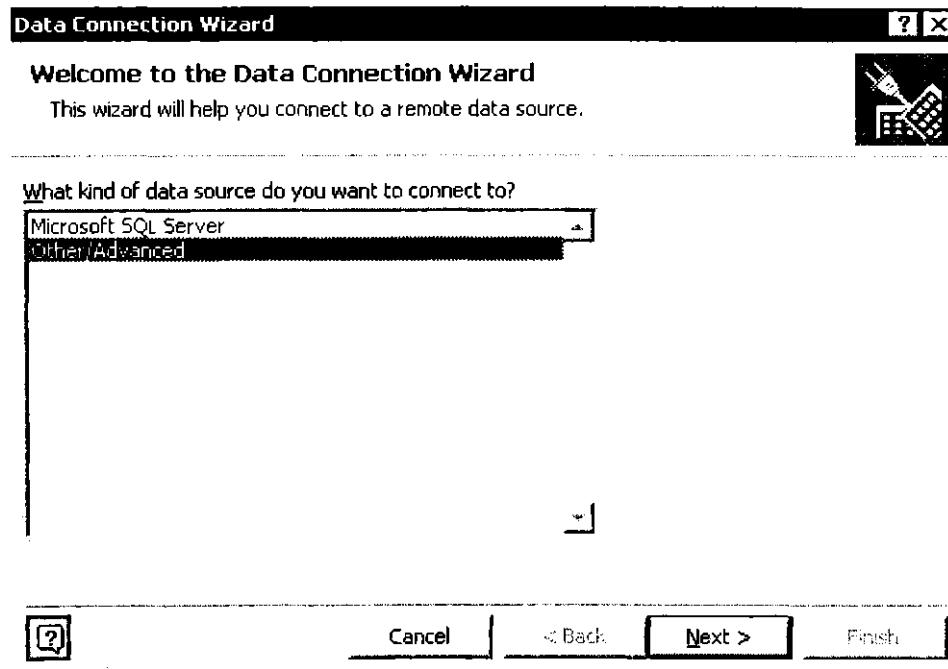


Рис. 10.40. Окно установления соединения с источником данных

Следующее окно мастера (рис. 10.41) предназначено для задания остальных свойств соединения, которые включают в себя: поставщика данных, имя сервера, информацию подключения (имя учетной записи пользователя и пароль), имя файла базы данных на сервере и другие.

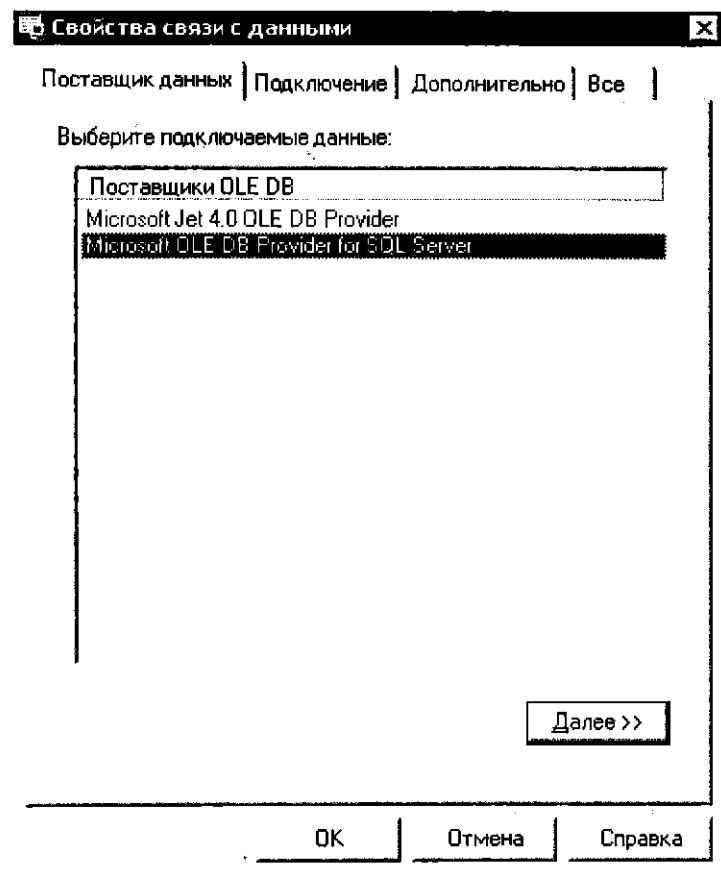


Рис. 10.41. Окно свойств связи с данными

Завершив установку параметров на вкладке подключения, рекомендуется проверить правильность параметров с помощью кнопки тестирования связи. Третья вкладка описывает параметры защиты данных и права доступа к ним.

Следующей и основной фазой разработки СДД является разработка собственно интерфейса СДД, которая по своей сути напоминает разработку форм или отчетов в режиме Конструктора (рис. 10.42). При этом используются аналогичные панели инструментов и приемы.

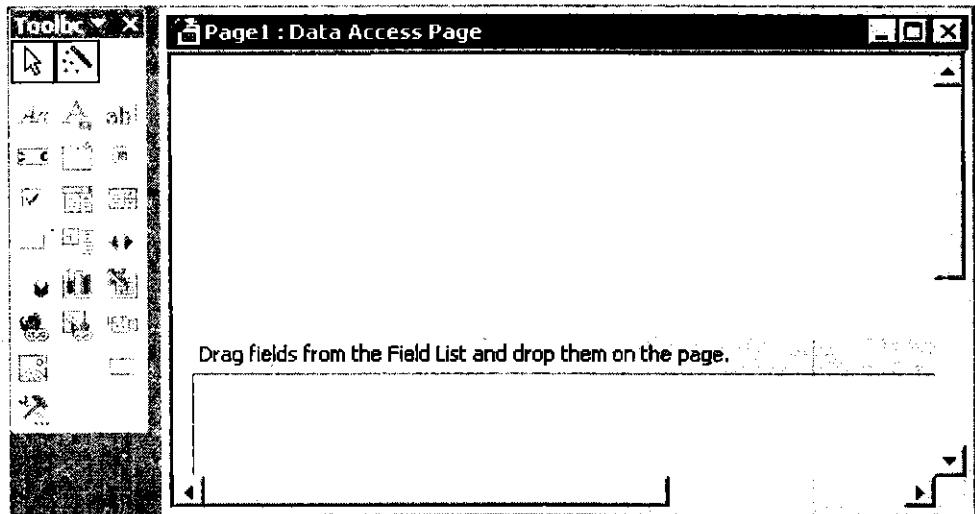


Рис. 10.42. Окно СДД в режиме конструктора (слева — панель инструментов)

Созданную таблицу сохраняют обычным способом сохранения объектов базы данных. Хранится СДД в отдельном htm-файле, который располагается в произвольном месте и имеет произвольное имя. Файл СДД логически связывается с базой данных таким образом, что он становится «виден» из окна базы данных (рис. 10.5).

Работа со страницами доступа к данным

Открыть и работать со СДД можно из среды Access и с помощью установленного браузера Интернета, например программы Microsoft Internet Explorer. Страницы можно использовать в приложении наряду с формами и отчетами. Создавать СДД, открывать в режиме Конструктора или просмотра в Access, а также просматривать и работать со страницами в Интернете или интранете можно при установке программы Microsoft Internet Explorer версии не ниже 5.

Зашитка страниц доступа к данным

Чтобы защитить страницу доступа к данным и данные, нужно выполнить следующее:

- защитить базу данных Access, содержащую ярлык (shortcut) страницы доступа к данным и соответствующий HTML-файл, используя средства защиты файловой системы;
- защитить базу данных, связанную со страницей доступа к данным, путем защиты базы данных от неправомочных пользователей с помощью

средств защиты на уровне пользователя. При этом можно управлять уровнем доступа в момент подключения пользователя;

- использовать многоуровневые средства обеспечения безопасности Microsoft Internet Explorer.

10.14. Разработка проекта

В Access 2002 понятие *проект* означает приложение пользователя, функционирующее в среде Access, но работающее с данными, хранящимися и обрабатываемыми на SQL-сервере. Заметим, что взаимодействие с данными SQL-сервера из Access 2002 можно произвести с помощью средств доступа ODBC. Это выполняется не из проекта, а из базы данных.

Общие сведения

Проекту Access 2002 соответствует новый тип файла (.adp), который работает как клиентское приложение SQL Server. Доступ к данным SQL-сервера осуществляется достаточно эффективно благодаря используемому интерфейсу OLE DB – OLE DB Provider for SQL Server (рис. 10.43). Проект содержит

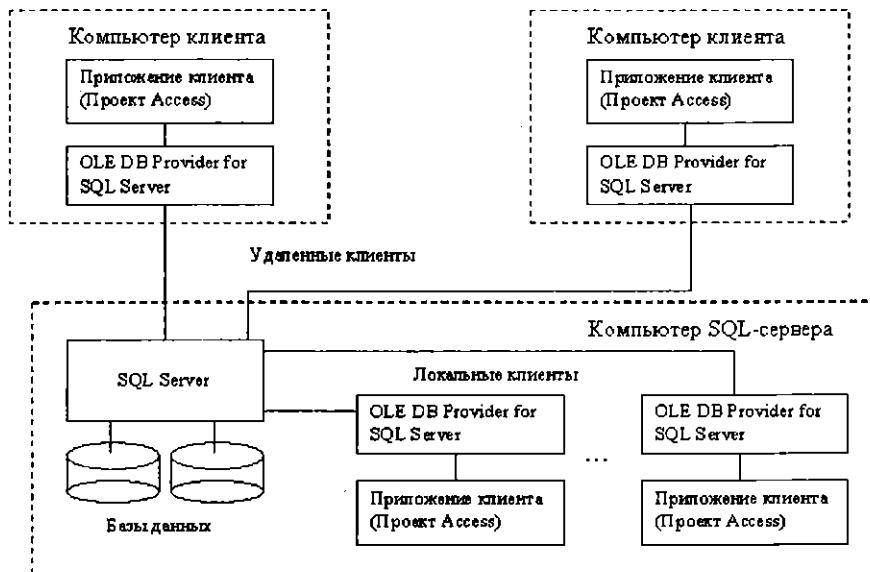


Рис. 10.43. Схема взаимодействия проекта Access и SQL-сервера

объекты, составляющие приложение: формы, отчеты, страницы доступа к данным, макросы и модули. В отличие от базы данных Access, проект не содержит данных и описаний основных объектов базы данных SQL-сервера: таблиц, представлений, схем баз данных и хранимых процедур. Эти объекты лишь отображаются в окне проекта Access. Проект может взаимодействовать с данными SQL-сервера, находясь на том же компьютере или на компьютере клиента. Компьютеры клиента и сервера в общем случае могут быть соединены с помощью сетевого адаптера, последовательного или параллельного кабеля, а также модема и телефонной линии связи.

Проект Access обеспечивает работу с существующей базой данных сервера и создание базы данных на SQL-сервере. Провайдер OLE DB (OLE DB Provider) позволяет отобразить в окне проекта (рис. 10.44) объекты базы данных SQL-сервера и предоставляет пользователю Access инструментальные средства интерактивной работы с данными SQL-сервера.

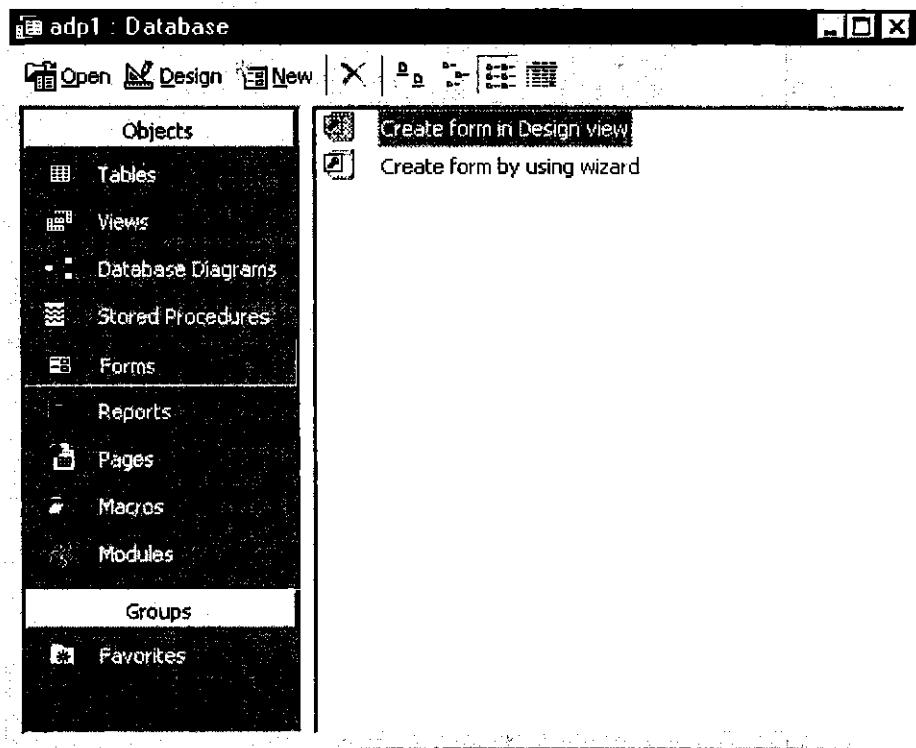


Рис. 10.44. Окно проекта Access

Объекты базы данных SQL-сервера используются в качестве источников данных в формах, отчетах, страницах доступа к данным, макросах и модулях

проекта, что дает возможность пользователю Access работать в привычном для него режиме. Объекты приложения создаются так же, как и в базе данных Access. Для быстрого создания простых приложений можно использовать Мастера.

Проект предоставляет возможность просматривать и корректировать данные, создавать и корректировать объекты SQL-сервера. Предоставляемые провайдером OLE DB инструментальные средства несколько отличаются от аналогичных средств Access, но освоить их несложно.

Через OLE DB осуществляется эффективный доступ не только к реляционным данным, но и ко многим другим типам источников данных в локальных и глобальных сетях, совместимых с OLE DB, таких как, например, файлы почты и электронные таблицы.

Проект Access может подключиться к базам данных следующих продуктов, установленных в системе Microsoft Windows NT, начиная с версии 4.0, или в системах Microsoft Windows 9x/2k:

- сервер Microsoft SQL Server 6.5 (с Server Service Pack 5 или более поздним), 7.0, 2000;
- инсталляция Microsoft Data Engine (MSDE);
- инсталляция Microsoft SQL Server 2000 Desktop Engine.

Создание проекта Access

Перед созданием проекта надо убедиться, что SQL-сервер, с которым предстоит работа, запущен, а база данных сервера доступна в соответствии с правами, определенными на сервере. При использовании Microsoft SQL Server требуется запустить программу Microsoft SQL Server Service Manager. Для установки параметров сервера и определения объектов, в том числе пользователей, используется программа SQL Server, открываемаяся при запуске программы сервера Enterprise Manager.

Если после запуска Access в начальном окне видна панель задач справа (рис. 10.1), то из панели можно сразу перейти к задаче создания проекта. Если же в начальном окне нет панели задач, увидеть ее можно с помощью команды **Файл | Создать (File | New)** или после нажатия кнопки **Создать (New)** панели инструментов.

В панели задач предлагается два варианта создания проекта: для работы с существующей на сервере базой данных — ссылка **Проект (существующая база данных) (Project (Existing Data))** или с одновременным созданием новой базы данных на сервере — ссылка **Проект (новая база данных) (Project (New Data))**. Коротко рассмотрим эти варианты.

1. Создание проекта для *существующей* на сервере базы данных. После выбора такого варианта создания проекта открывается окно **Файл новой базы данных (File New Database)**. В нем выбирается местоположе-

ние сохраняемого файла проекта и его имя. Для завершения работы с окном нажимается кнопка **Создать (Create)**.

При этом открывается окно **Свойства связи с сервером (Data Link Properties)** (рис. 10.45), на вкладке **Соединение (Connection)** которого вводится имя сервера (пункт 1), информация о подключении к серверу (пункт 2) и имя базы данных на сервере (пункт 3). Дополнительные параметры подключения, в частности величину задержки (таймаута) в секундах при обмене данными с сервером, можно задать на вкладке **Дополнительно (Advanced)**. Введенные параметры сохраняются в проекте, поэтому при повторном открытии проекта их вводить не нужно.

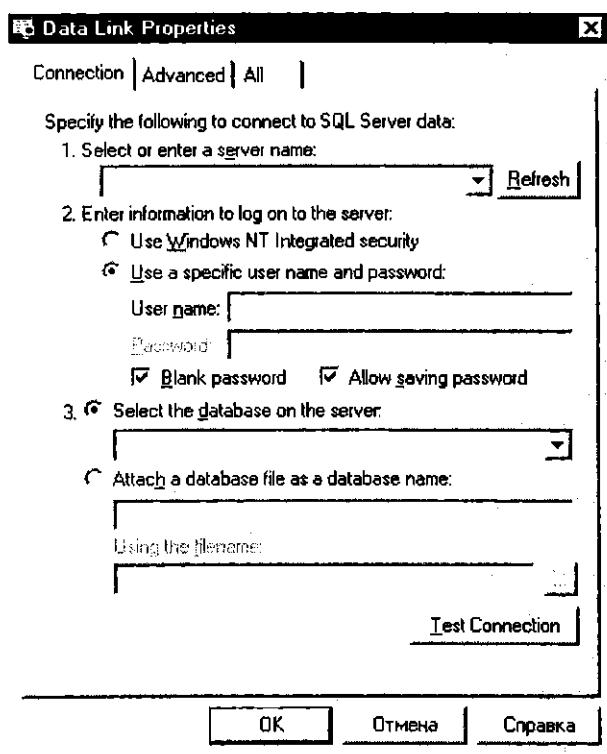


Рис. 10.45. Окно свойств связи с сервером

После ввода параметров полезно протестировать описанную связь нажатием кнопки **Test Connection** (тест соединения). Если тестирование прошло успешно, нужно нажать **OK**, что приводит к появлению окна проекта, подобного показанному на рис. 10.44. В нем создаются объекты приложения.

После создания проекта в меню **Файл** (File) становится доступной команда **Подключение** (Connection), которая открывает окно **Data Link Properties** (рис. 10.45). Это позволяет проверить или переопределить параметры подключения к серверу, выбрать базу данных сервера, с объектами которой можно работать в окне проекта Access, и установить связь с другим сервером.

2. Создание проекта и новой базы данных на сервере. После выбора варианта, по которому создание проекта сопровождается созданием на сервере базы данных, открывается окно **Файл новой базы данных** (File New Database). В нем выбирается местоположение сохраняемого файла проекта и его имя. После завершения работы с окном нажимается кнопка **Создать** (Create). При этом открывается окно базы данных и запускается **Мастер баз данных SQL-сервера** (Microsoft SQL Server Database Wizard). В окне Мастера (рис. 10.46) указывается сервер, с которым надо установить соединение, имя пользователя, пароль, и имя создаваемой на сервере базы данных. Зарегистрированный на сервере пользователь должен иметь права, достаточные для создания на сервере базы данных.

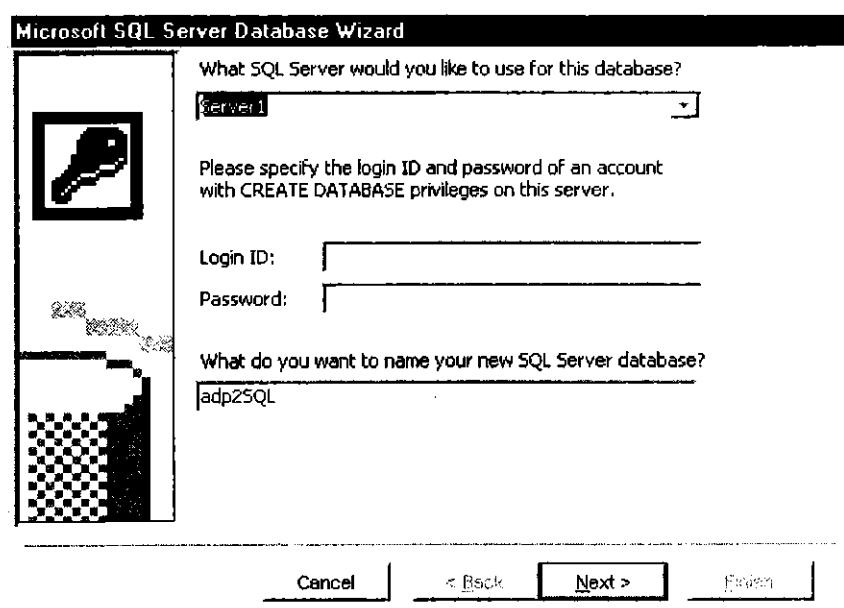


Рис. 10.46. Окно Мастера базы данных на SQL-сервере

Далее Мастер сообщает о начале создания на сервере базы данных, предупреждая о том, что операция может занять некоторое время. Мастер создает на сервере пустую базу данных и добавляет новый объект в папку /Database

сервера. По умолчанию создаваемой базе данных присваивается имя проекта Access, дополненное символами SQL.

После создания базы данных Мастером открывается окно нового проекта, в котором выполняется создание объектов базы данных и приложения.

В результате создания проекта и новой базы данных на сервере на компьютере клиента создается файл проекта с расширением adp, а на сервере – файлы с расширениями mdf и ldf.

Контрольные вопросы и задания

1. Дайте общую характеристику СУБД Access 2002.
2. Охарактеризуйте новые возможности Access 2002 по сравнению с предыдущими версиями.
3. Дайте определение основных элементов базы данных Access.
4. Изобразите схему взаимосвязи основных объектов БД Access.
5. Охарактеризуйте средства поддержки проектирования в СУБД Access.
6. Опишите технологию создания базы данных Access.
7. Каким образом можно создавать таблицы базы данных Access?
8. Каким образом осуществляется связывание таблиц?
9. Перечислите варианты и опишите технологию создания запросов.
10. Как осуществляется создание форм?
11. Охарактеризуйте варианты создания отчетов.
12. Что представляют собой макросы и как они создаются?
13. Охарактеризуйте назначение и технологию создания модулей.
14. Укажите особенности построения SQL-запросов.
15. Охарактеризуйте связь языков QBE и SQL.
16. Охарактеризуйте технологию применения SQL в запросах и отчетах.
17. Каким образом SQL используется в макросах?
18. Как SQL используется в программах на VBA?
19. Охарактеризуйте способы защиты баз данных Access.
20. Как осуществляется защита на уровне пользователя?
21. Для чего создаются рабочие группы?
22. Охарактеризуйте типы прав доступа.
23. Как осуществляется определение прав пользователей и групп?
24. Охарактеризуйте шифрование баз данных Access.
25. Для чего применяется скрытие объектов?
26. Охарактеризуйте способы обслуживания баз данных.
27. Что представляет собой репликация баз данных и как она осуществляется?
28. Как создаются и удаляются реплики?
29. Что означает синхронизация реплик?
30. Охарактеризуйте технологию работы с мультимедиа-данными.

31. Назовите средства повышения производительности СУБД Access.
32. Охарактеризуйте средства работы в Internet и intranet.
33. Перечислите средства помощнико и трассировки.
34. Укажите особенности работы с данными.
35. Назовите средства просмотра и поиска.
36. Назовите новые сервисные средства работы с БД.
37. Как осуществляется настройка и оптимизация выполнения запросов?
38. Укажите новые возможности работы в окнах Access.
39. Назовите новые возможности для разработки программ на языке VBA.
40. Для чего и как создаются mde-файлы приложений?
41. Опишите технику создания гиперссылок.
42. Как выполняется редактирование гиперссылок?
43. Что представляют собой страницы доступа к данным и как они создаются?
44. Охарактеризуйте понятие проекта в Access.
45. Какова схема взаимодействия проекта в Access и SQL-сервера?
46. Опишите технологию создания проекта Access.

Варианты индивидуального задания

Индивидуальное задание включает следующие элементы:

- проектирование БД;
- создание БД средствами Access и занесение в нее данных;
- организацию запросов к базе;
- оформление отчета с помощью Конструктора запросов.

Обучаемому необходимо спроектировать БД, содержащую некоторые сведения, представленные в виде группы представленных ниже атрибутов. Шесть первых атрибутов являются обязательными для всех, а остальные варьируются (табл. 10.3). Приведенные атрибуты характеризуют некоторую группу людей и позволяют с учетом их профессиональной деятельности рассчитать денежное содержание. Состав атрибутов:

1. FIO — фамилия и инициалы;
2. God — год рождения;
3. Dolgn — должность занимаемая;
4. O_Dolgn — оклад по должности;
5. Ctag — стаж работы;
6. D_Stag — надбавка за стаж (свыше 5, 10, 15, 20, 25, 30, 35, 40 лет);
7. Udal — удаленность (средняя, большая, очень большая);
8. D_Udal — надбавка за удаленность;
9. Slogn — сложность (средняя, высокая, очень высокая);
10. D_Slogn — надбавка за сложность;

11. Vredn – вредность (по категориям: 1, 2, 3, 4, 5);
12. D_Vredn – надбавка за вредность;
13. Clasn – классность (мастер, первая, вторая, третья);
14. D_Clasn – надбавка за классность;
15. U_Zvan – ученое звание (доцент, снс);
16. D_Uzvan – надбавка за ученое звание;
17. U_Step – ученая степень (ктн, дтн);
18. D_UStep – надбавка за ученую степень;
19. Zvanie – воинское звание, офицерские (младший лейтенант, лейтенант, ст. лейтенант, капитан, майор, подполковник, полковник);
20. D_Zvanie – надбавка за воинское звание.

Таблица 10.3
Распределение атрибутов по вариантам

Вариант	Номер атрибута											
	9	10	11	12	13	14	15	16	17	18	19	20
1	*	*									*	*
2	*	*							*	*		
3	*	*					*	*				
4	*	*			*	*						
5	*	*	*	*								
6			*	*							*	*
7		*	*						*	*		
8		*	*				*	*				
9		*	*	*	*							
10				*	*						*	*
11									*	*		
12			*	*	*	*						
13						*	*				*	*
14						*	*	*	*			
15								*	*	*	*	*
16	*	*						*	*	*	*	*
17	*	*				*	*			*	*	*
18	*	*					*	*	*	*		
19	*	*			*	*					*	*
20	*	*	*	*							*	*
21	*	*			*	*	*	*				
22	*	*	*	*	*	*						
23		*	*					*	*	*	*	*
24		*	*			*	*	*	*			
25		*	*	*	*	*	*	*				
26			*	*	*	*	*	*			*	*
27				*	*	*	*	*	*	*		
28						*	*	*	*	*	*	*

Задание каждому обучаемому состоит в следующем.

1. Необходимо в соответствии со своим списком атрибутов спроектировать БД.
2. Создать базу данных.
3. Занести в нее данные.
4. Организовать постоянные связи между таблицами для обеспечения целостности своей БД при: изменении записей, добавлении записей, удалении записей.
5. Убедиться, что:
 - данные, внесенные в таблицы, непротиворечивы;
 - система поддержки целостности БД функционирует. Для этого попытаться изменить, ввести и удалить данные в таблицах с нарушением правил поддержания целостности БД.
6. Организовать запросы к БД, которые позволяли бы продемонстрировать:
 - фамилию и должность сотрудника;
 - сумму денежного содержания сотрудника и значения компонентов, из которых она формируется;
 - для каждого поля сформировать заголовок, используя кириллицу;
 - сведения в запросе упорядочить в порядке убывания денежного содержания, а при равном денежном содержании — в алфавитном порядке фамилий сотрудников.
7. Оформить отчет, используя Конструктор отчетов.

Литература

1. Вейскас Д. Эффективная работа с Microsoft Access 97. — СПб.: Питер Ком, 1999. — 976 с.
2. Дженнингс Р. Microsoft Access 97 в подлиннике. Том II / Пер. с англ. — СПб.: БНВ—Санкт-Петербург, 1997. — 688 с.
3. Мэри Кэмбелл. Access. Ответы / Пер. с англ. — М.: Восточная Книжная Компания, 1996. — 336 с.
4. Бекаревич Ю.Б., Пушкина Н. В. Microsoft Access 2000. — СПб.: БХВ—Санкт-Петербург, 1999. — 480 с.
5. Михеева В., Харитонова И. Microsoft Access 2002. — СПб.: БХВ-Петербург, 2002. — 1040 с.
6. Михеева В., Харитонова И. Microsoft Access 2003. Наиболее полное руководство. — СПб.: БХВ-Петербург, 2004.
7. Хомоненко А. Д., Гридин В. В. Microsoft Access. Быстрый старт. — СПб.: БХВ-Петербург, 2002.

11. Borland C++ Builder

Прикладные программы, или приложения, C++ Builder (обычные и для работы с базами данных) создаются в интегрированной среде разработки (IDE – Integrated Development Environment). Пользовательский интерфейс этой среды служит для организации взаимодействия с программистом и включает в себя ряд окон, содержащих различные элементы управления. С помощью средств интегрированной среды разработчику удобно проектировать интерфейсную часть приложения, а также писать программный код и связывать его с элементами управления. В интегрированной среде разработки C++ Builder проходят все этапы создания приложения, включая отладку.

11.1. Пользовательский интерфейс

Интегрированная среда разработки C++ Builder представляет собой многооконную систему. Вид интегрированной среды разработки (пользовательский интерфейс) может различаться в зависимости от настроек. После загрузки интерфейс C++ Builder выглядит так, как показано на рис. 11.1 и первоначально включает шесть окон:

- Главное окно (C++ Builder — Project1);
- окно Обозревателя дерева объектов (Object TreeView);
- окно Инспектора объектов (Object Inspector);
- окно Формы, или Конструктора формы (Form1);
- окно Редактора кода (Unit1.cpp);
- окно Проводника класса (ClassExplorer).

Последние два окна находятся позади окна Формы, причем окно Проводника класса пристыковано слева к окну Редактора кода, поэтому оба этих окна имеют общий заголовок **Unit1.cpp**.

На экране кроме указанных окон могут присутствовать и другие окна, отображаемые при вызове соответствующих средств, например, окно Редактора изображений (Image Editor). Окна C++ Builder можно перемещать, изменять их размеры и убирать с экрана (кроме главного окна), а также состыковывать между собой.

Несмотря на наличие многих окон, C++ Builder является однодокументной средой и позволяет одновременно работать только с одним приложением (проектом приложения). Название проекта приложения выводится в строке заголовка главного окна в верхней части экрана.

При сворачивании главного окна сворачивается весь интерфейс C++ Builder и, соответственно, все открытые окна; при закрытии главного окна работа с C++ Builder прекращается. Главное окно C++ Builder включает:

- главное меню;
- панели инструментов;
- палитру компонентов.

Главное меню содержит обширный набор команд для доступа к функциям C++ Builder, основные из которых рассматриваются при изучении связанных с этими командами операций.

Панели инструментов находятся под главным меню в левой части главного окна и содержат пятнадцать кнопок для вызова наиболее часто используемых команд главного меню, например, File\Open (Файл\Открыть) или Run\Run (Выполнение\Выполнить).

Вызвать многие команды главного меню можно также с помощью комбинаций клавиш, указываемых справа от названия соответствующей команды. Например, команду Run\Run можно вызвать с помощью клавиши <F9>, а команду View\Units (Просмотр\Модули) — с помощью комбинации клавиш <Ctrl>+<F12>.

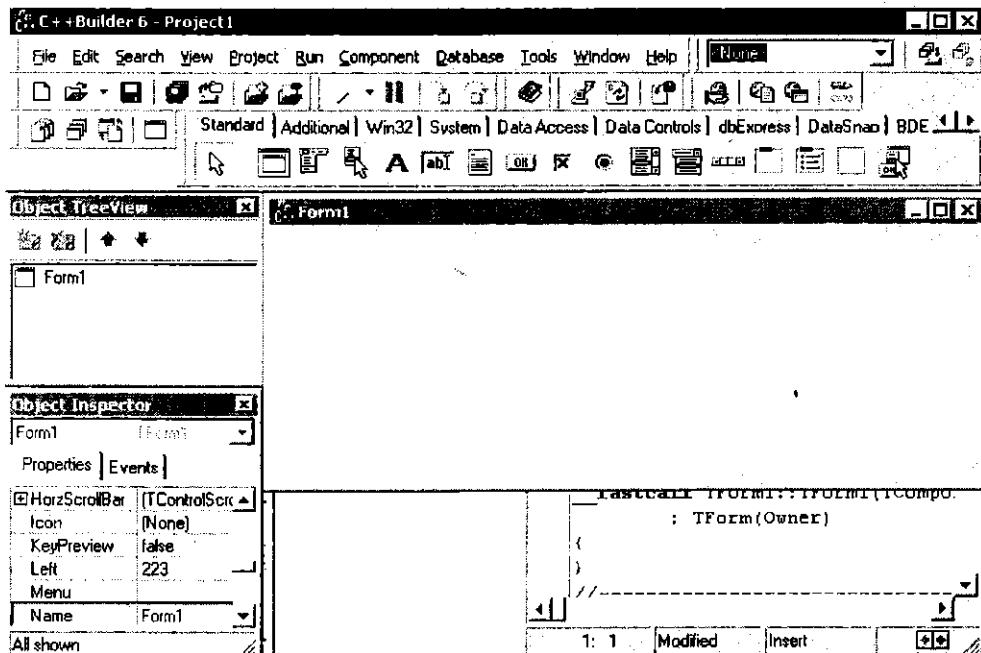


Рис. 11.1. Вид интегрированной среды разработки

Всего имеется 7 панелей инструментов:

- Standard (Стандартная);
- View (Просмотр);
- Debug (Отладка);
- Custom (Пользователь);
- Desktop (Рабочий стол);
- CORBA;
- Internet (Интернет).

Отображением панелей инструментов и настройкой кнопок на них можно управлять. Эти действия выполняются с помощью контекстного меню панелей инструментов, вызываемого щелчком правой кнопки мыши при размещении указателя в области панелей инструментов или главного меню.

С помощью контекстного меню можно также управлять видимостью Палитры компонентов (Component Palette).

Палитра компонентов находится под главным меню в правой части главного окна и содержит множество компонентов, размещаемых в создаваемых формах. Компоненты являются своего рода строительными блоками, из которых конструируются формы приложения. Все компоненты разбиты на группы, каждая из которых в Палитре компонентов располагается на отдельной странице, а сами компоненты представлены значками. Нужная страница Палитры компонентов выбирается щелчком мыши на ее значке. В составе Палитры компонентов к числу важнейших можно отнести следующие страницы:

- Standard – стандартная;
- Additional – дополнительная;
- Win32 – 32-разрядного интерфейса Windows;
- System – доступа к системным функциям;
- Data Access – работы с информацией из баз данных;
- Data Controls – создания элементов управления данными;
- dbExpress – доступа к SQL-серверам;
- DataSnap – создания многоуровневых приложений баз данных;
- BDE – доступа к данным с помощью BDE;
- Internet – создания приложений Internet;
- QReport – генерации отчетов в приложении;
- Dialogs – стандартных диалогов.

Окно Формы (или Конструктора формы) первоначально находится в центре экрана и имеет заголовок **Form1**. В нем выполняется проектирование формы, в процессе которого в форму из Палитры компонентов помещаются необходимые компоненты. При этом проектирование заключается в визуальном конструировании формы, а действия разработчика похожи на работу в среде простого графического редактора. Поскольку при проектировании разработ-

чик имеет дело непосредственно с формой, часто окно Конструктора формы также называют окном Формы или просто «формой».

Окно Редактора кода (*Unit1.cpp*) после запуска системы программирования находится под окном Формы и почти полностью перекрывается им. Редактор кода представляет собой обычный текстовый редактор, с помощью которого можно редактировать текст модуля и другие текстовые файлы приложения, например, файл проекта. Каждый редактируемый файл находится в окне Редактора кода на отдельной странице, доступ к которой осуществляется щелчком на соответствующем значке. Первоначально в окне Редактора кода на странице *Code* содержится одна вкладка *Unit1* исходного кода модуля формы *Form1* разрабатываемого приложения.

Переключаться между окнами Формы и Редактора кода удобно с помощью клавиши <F12>.

Окно Проводника класса (*ClassExplorer*) пристыковано слева к окну Редактора кода. В нем в виде дерева отображаются все объекты модуля формы, например переменные и процедуры (рис. 11.2). В окне Проводника класса удобно просматривать объекты приложения и быстро переходить к нужным объектам, что особенно важно для больших модулей. Окно Проводника класса открывается командой *ClassExplorer\View* (Просмотр\Проводник класса).

Окно Обозревателя дерева объектов (рис. 11.3) после запуска системы находится под Главным окном и отображает древовидную структуру объектов текущей формы (первоначально *Form1*). Его можно открыть командой *View\Object TreeView* (Просмотр/Просмотр дерева объектов).

Окно Инспектора объектов (рис. 11.1) находится под окном Обозревателя дерева объектов в левой части экрана и отображает свойства и события объектов для текущей формы *Form1*. Его можно открыть командой *View\Object Inspector* (Просмотр\Инспектор объектов) или нажатием клавиши <F11>.

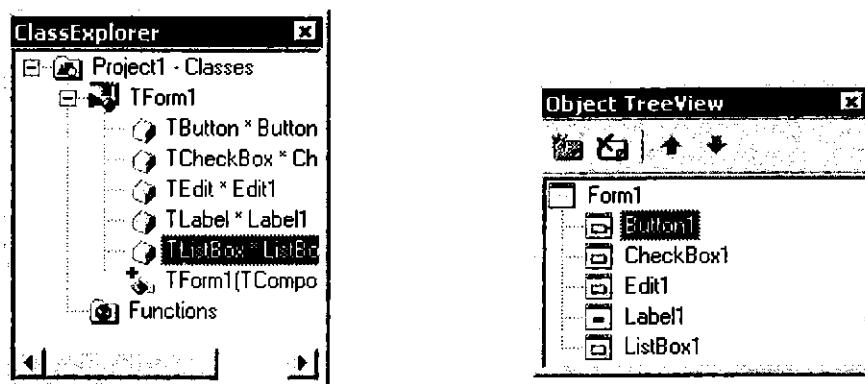


Рис. 11.2. Окно Проводника класса

Рис. 11.3. Окно Обозревателя дерева объектов

Окно Инспектора объектов имеет две страницы: **Properties (Свойства)** и **Events (События)**.

Страница **Properties** отображает информацию о текущем (выбранном) компоненте в окне Формы и при проектировании формы позволяет удобно изменять многие свойства компонентов.

Страница **Events** определяет процедуры обработки различных событий для выбранного компонента. Если для какого-либо события задана такая процедура, то в процессе выполнения приложения при возникновении этого события процедура вызывается автоматически. Такие процедуры служат для обработки соответствующих событий, поэтому их называют *процедурами – обработчиками событий* или просто *обработчиками*. Отметим, что события также являются свойствами, которые указывают на свои обработчики.

В конкретный момент времени Инспектор объектов отображает свойства и события текущего (выбранного) компонента, имя и тип которого отображаются в списке под заголовком окна Инспектора объектов. Компонент, расположенный в форме, можно выбрать щелчком мыши на нем или выбором в списке Инспектора объектов. У каждого компонента есть набор свойств и событий, определяющих его особенности.

Инспектор объектов позволяет группировать свойства и события по категориям или в алфавитном порядке. Свойства (и их значения) отображаются различными цветами. В Инспекторе объектов содержатся и свойства, предназначенные только для чтения.

По умолчанию Инспектор объектов отображает названия свойств и событий в алфавитном порядке (см. рис. 11.1). Отображение их по категориям выполняется командой **Arrange\by Category** (Расположить\По категориям) контекстного меню Инспектора объектов.

По умолчанию Инспектор объектов отображает все свойства и события объектов. Можно отключить/включить отображение некоторой категории, убрав/установив отметку в соответствующем пункте (например, **Action**) подменю команды **View** контекстного меню.

C++ Builder поддерживает технологию Dock-окон, которые могут стыковаться (соединяться) друг с другом с помощью мыши. Такими окнами являются инструментальные (недиалоговые) окна интегрированной среды разработки, в том числе окна Инспектора объектов и Проводника кода. Состыкованные окна удобно, например, перемещать по экрану или изменять их размеры.

Для соединения двух окон следует с помощью мыши поместить одно из них на другое и после изменения вида рамки перемещаемого окна отпустить его, после чего это окно автоматически пристыкуется сбоку ко второму окну. Разделение окон выполняется перемещением пристыкованного окна за двойную линию, размещенную под общим заголовком. После соединения окна представляют собой одно общее окно, разделенное на несколько частей. При

стыковке/отстыковке окно изменяет свое название. Так, окно Проводника кода, состыкованное с окном Редактора класса, имеет общее с ним название, например, *Unit1.cpp*, при отстыковке название его сохраняется. Окна Инспектора объектов и Обозревателя дерева объектов при стыковке объединяют свои названия (названия всех окон указываются через запятую).

Для окон Инспектора объектов и Обозревателя дерева объектов можно установить режим *Stay on Top* (Расположить наверху), расположив их поверх других окон. Это выполняется включением одноименной отметки в контекстном меню.

11.2. Характеристика проекта

В этом разделе рассматриваются: состав проекта, файл проекта, файлы формы, файлы модулей, файл ресурсов и параметры проекта.

Состав проекта

Создаваемое в среде C++ Builder приложение состоит из нескольких элементов, объединенных в проект. В состав проекта входят следующие элементы (в скобках указаны расширения имен файлов):

- исходный код проекта (*cpr*);
- исходный код модулей форм и модулей (*cpr*);
- заголовочные файлы модулей форм и модулей (*h*);
- представления форм (*dfm* – для Windows, *xfm* – кроссплатформенный вариант);
- параметры проекта (*bprg*);
- параметры среды (*cfg*);
- описание ресурсов (*res*).

Взаимосвязи между отдельными частями (файлами) проекта показаны на рис. 11.4.

Кроме приведенных файлов, автоматически могут создаваться и другие файлы, например, резервные копии файлов: *~dp* – для файлов с расширением *dpr*; *~cp* – для файлов с расширением *cp*; *~h* – для файлов с расширением *.h*. При создании группы проектов создается текстовый файл с расширением *.bpg*; при работе с пакетами создаются двоичные файлы с расширениями *.bpl* и *.bpi*.

При запуске C++ Builder автоматически создается новый проект с именем *Project1*, отображаемым в заголовке главного окна C++ Builder. Этот проект имеет в своем составе одну форму *Form1*, название которой видно в окне Формы. Разработчик может изменить предлагаемое по умолчанию имя проекта, а также установить параметры среды таким образом, что после загрузки

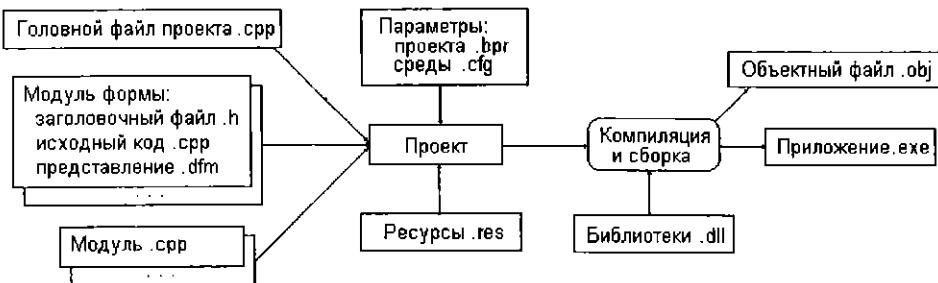


Рис. 11.4. Связи между файлами проекта

C++ Builder будет автоматически загружаться приложение, разработка которого выполнялась в последний раз.

Обычно файлы проекта располагаются в одном каталоге. Поскольку даже относительно простой проект включает в себя достаточно много файлов, а при добавлении к проекту новых форм количество этих файлов увеличивается, для каждого нового проекта целесообразно создавать отдельный каталог, где и сохранять все файлы проекта.

Головной файл проекта

Головной файл проекта является центральным файлом проекта, имеет расширение .cpp и представляет собой собственно программу. Для приложения, имеющего в составе одну форму, головной файл проекта имеет следующий вид:

```

//-----
// Директивы препроцессора
#include <vcl.h>
#pragma hdrstop
//-----
// Макросы
USEFILE("readme.txt");
USEFORM("main.cpp", FormMain);
USERES("scrollba.res");
//-----
// Главная функция WinMain
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TFormMain), &FormMain);
        Application->Run();
    }
}
  
```

```
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
//-----
```

Имя проекта (программы) совпадает с именем головного файла проекта и указывается при сохранении этого файла на диске, первоначально это имя **Project1**. То же имя имеют файлы ресурсов и параметров проекта, при переименовании файла проекта данные файлы переименовываются автоматически.

Сборка всего проекта выполняется при компиляции файла проекта. При этом имя создаваемого приложения (exe-файл) или динамически загружаемой библиотеки (dll-файл) совпадает с названием файла проекта. В дальнейшем мы будем подразумевать, что создается приложение, а не динамически загружаемая библиотека.

В начале головного файла проекта содержатся различные директивы препроцессора. В частности с помощью директивы `#include <vcl.h>` выполняется подключение заголовочного файла с объявлениями, используемыми в библиотеке визуальных компонентов.

Далее следуют макросы, с помощью которых выполняется подключение файлов ресурсов и форм. В частности, макрос `USEFORM("Main.cpp", FormMain)` выполняет подключение модуля формы, расположенной в файле с именем **Main.cpp** и имеющей имя **FormMain**. В свою очередь, макрос `USERES("scrollba.res")` выполняет подключение файла ресурсов **scrollba.res**.

Далее расположена главная функция приложения – `WinMain`. Первый и второй параметры этой функции являются лескапторами данного экземпляра приложения и предыдущего экземпляра приложения (если одновременно выполняется несколько таких приложений). Третий параметр служит указателем на строку с параметрами, передаваемыми приложению через командную строку.

Программа проекта содержит три оператора, выполняющих инициализацию приложения (`Application->Initialize();`), создание формы (`Application->CreateForm(__classid(TFormMain), &FormMain);`) **Form1** и запуск приложения (`Application->Run();`). Названные операторы размещены в блоке `try`, за которым следует блок `catch`. Сделано это с целью обработки исключений, возможных при выполнении приложения. При этом используется стандартный обработчик исключений с применением функции `ShowException`.

При выполнении разработчиком каких-либо операций с проектом C++ Builder формирует код файла проекта автоматически. Например, при добав-

лении новой формы в файл проекта добавляются две строки кода, относящиеся к этой форме, а при исключении формы из проекта эти строки автоматически исключаются. При необходимости программист может вносить изменения в файл проекта самостоятельно, однако подобные действия могут разрушить целостность проекта и поэтому обычно выполняются только опытными программистами. Отметим, что некоторые операции, например, создание обработчика события для объекта Application, системой C++ Builder автоматически не выполняются и требуют самостоятельного кодирования в файле проекта.

Отображение кода головного файла проекта в окне Редактора кода задается командой Project\View Source (Проект\Просмотр источника).

В файле проекта для многих приложений имеется похожий код, поэтому в дальнейшем при рассмотрении большинства приложений содержимое этого файла нами не приводится.

Файлы формы

Для каждой формы в составе проекта автоматически создаются файл представления формы (расширение .dfm) и файлы модуля формы: заголовочный файл (расширение .h) и файл исходного кода формы (расширение .cpp).

Файл представления формы является ресурсом C++ Builder и содержит характеристики формы и ее компонентов. Разработчик обычно управляет этим файлом через окна Формы и Инспектора объектов. При конструировании формы в файл описания автоматически вносятся соответствующие изменения. Содержимое файла представления формы определяет ее вид. При необходимости можно отобразить этот файл на экране в текстовом виде, что выполняется командой View as Text (Просмотреть как текст) контекстного меню формы. При этом окно Формы пропадает с экрана, а содержимое файла представления формы открывается в окне Редактора кода и доступно для просмотра и редактирования. Например, файл представления формы, включающей кнопку (*object Button1*) и односторонний редактор (*object Edit1*), содержит следующий текст:

```
object Form1: TForm1
  Left = 192
  Top = 133
  Width = 544
  Height = 375
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
```

```
Font.Style = []
OldCreateOrder = False
PixelsPerInch = 96
TextHeight = 13
object Button1: TButton
  Left = 144
  Top = 40
  Width = 57
  Height = 49
  Caption = 'Button1'
  TabOrder = 0
end
object Edit1: TEdit
  Left = 224
  Top = 136
  Width = 57
  Height = 21
  TabOrder = 1
  Text = 'Edit1'
end
end
```

Чтобы вернуться обратно от просмотра текстового вида файла представления к просмотру внешнего вида формы, нужно выполнить команду **View as Form** (Просмотреть как форму) также контекстного меню формы.

Файл представления содержит перечень всех объектов формы, включая саму форму, а также свойства этих объектов. Для каждого объекта указывается его тип; для формы ее тип (класс) **TForm1** описывается в модуле этой формы. Если в строчке **Caption = 'Form1'**, определяющей заголовок формы, вместо **Form1** ввести, например, текст **Первая форма**, то заголовок формы изменится на новый. Однако на практике подобные действия обычно выполняются в окне Инспектора объектов.

Повторное открытие окна формы выполняется командой **View\Forms** (Просмотр\Формы) или комбинацией клавиш **<Shift>+<F12>**, после чего открывается диалоговое окно **View Form** (Просмотр форм), в списке которого и выбирается нужная форма.

Одновременно можно отобразить на экране несколько форм. Для закрытия того или иного окна Формы нужно выполнить команду **File\Close** (Файл\Закрыть) или щелкнуть мышью на кнопке закрытия соответствующего окна.

Заголовочный файл модуля формы содержит описание класса формы. Например, для формы, на которой размещены кнопка (**Button1**), надпись (**Label1**) и односторонний редактор (**Edit1**), заголовочный файл модуля формы содержит следующий код:

```
//-----
#ifndef Unit1H
#define Unit1H
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//-----
class TForm1 : public TForm
{
public: // IDE-managed Components
    TButton *Button1;
    TLabel *Label1;
    TEdit *Edit1;
    void __fastcall Button1Click(TObject *Sender);
private: // User declarations
public: // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif
```

В начале заголовочного файла содержатся директивы препроцессора, подключаемые автоматически. При необходимости соответствующие директивы разработчик может добавлять вручную. Далее следует описание класса формы (в нашем примере `TForm1`). В разделе `_published` содержатся автоматически добавляемые объявления помещенных в форму компонентов (`Button1`, `Label1` и `Edit1`) и обработчиков событий для них (`Button1Click`). В разделах `private` (собственный) и `public` (общедоступный) разработчик может поместить свои объявления типов, переменных и функций. Кроме того, в разделе `public` находится автоматически включенный прототип конструктора формы (`__fastcall TForm1(TComponent* Owner);`). Объявления, размещенные в разделе `private`, доступны только в пределах данного модуля; объявления, размещенные в разделе `public`, доступны для других классов и модулей. Далее размещено автоматически подключенное предложение `PACKAGE`, которое выходит за рамки нашего рассмотрения.

Файл *исходного кода формы*, называемый также *файлом реализации*, содержит директивы препроцессора, объявления типов и переменных, а также исходный код обработчиков событий, которые определяют функциональность приложения. Файл реализации, к примеру, может иметь следующий вид:

```
//-----  
// Директивы препроцессора  
#include <vcl.h>  
#pragma hdrstop  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
// Объявление формы как объекта  
TForm1 *Form1;  
//-----  
// Вызов конструктора формы  
__fastcall TForm1::TForm1(TComponent* Owner)  
: TForm(Owner)  
{  
}  
// Место размещения объявлений типов  
// и переменных, описаний функций  
//-----  
// Обработчик события нажатия кнопки Button1  
void __fastcall TForm1::Button1Click(TObject *Sender)  
{  
Edit1->Text = "0";  
}  
//-----
```

В начале файла реализации модуля формы содержатся автоматически включаемые директивы препроцессора. При необходимости разработчик может добавлять свои директивы препроцессора, к примеру, для подключения других модулей. В тело функции вызова конструктора формы можно добавлять операторы, реализуемые при создании формы. Впрочем, часто для этих целей создается обработчик события **OnCreate** для формы.

C++ Builder автоматически создает файлы модуля формы при добавлении новой формы. По умолчанию к проекту добавляется новая форма типа TForm, не содержащая компонентов.

При размещении в форме компонентов, а также при создании обработчиков событий в модуль формы вносятся соответствующие изменения. При этом часть этих изменений C++ Builder выполняет автоматически, а часть пишет разработчик. Обычно все действия разработчика, связанные с программированием, выполняются именно в модулях форм.

Тексты файлов модулей форм отображаются и редактируются с помощью Редактора кода. Открыть файл модуля формы можно в стандартном окне открытия файла (команда **File\Open** (Файл\Открыть)) или в диалоговом окне

View Unit, открываемом командой **View\Units** (Просмотр\Модули) или нажатием комбинации клавиш **<Ctrl>+<F12>**. В окне открытия файла модуля формы можно выбрать также файл проекта. После выбора нужного модуля (или проекта) и нажатия кнопки **OK** его текст появляется на отдельной странице Редактора кода.

Если выбран модуль формы, то в окне Редактора кода первоначально отображается содержимое файла реализации формы (например, **Unit1.cpp**). Чтобы перейти к отображению содержимого заголовочного файла этого же модуля формы, достаточно в нижней части окна Редактора кода выбрать вкладку с названием **Unit1.h** (см. рис. 11.5).

Отметим, что файлы каждой формы (представления и модуля) имеют одинаковые имена, отличные от имени файла проекта.

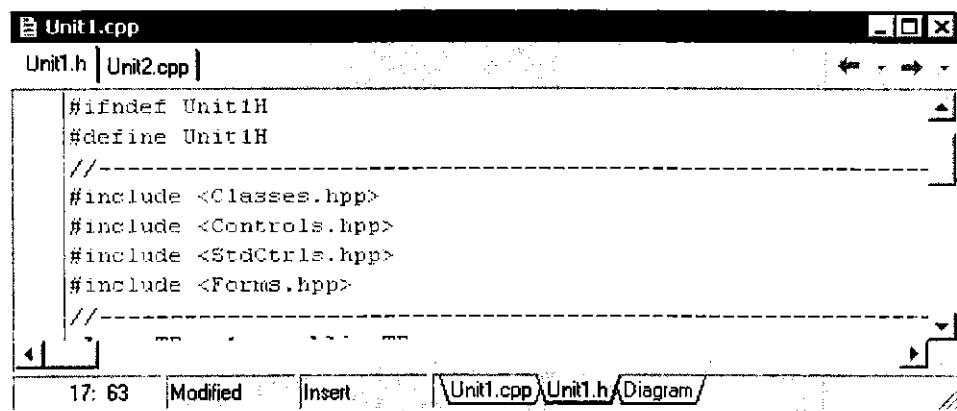


Рис. 11.5. Вид заголовочного файла модуля формы в окне Редактора кода

Файлы модулей

Кроме модулей в составе форм, при программировании можно использовать и *отдельные модули*, не связанные с какой-либо формой. Они оформляются по обычным правилам языка C++ и сохраняются в отдельных файлах. Для подключения модуля его имя указывается в директиве препроцессора **#include** того модуля или проекта, который использует средства этого модуля.

В отдельном модуле можно и полезно размещать процедуры, функции, константы и переменные, общие для нескольких модулей проекта.

Файл ресурсов

При первом сохранении проекта автоматически создается файл ресурсов (расширение .res) с именем, совпадающим с именем файла проекта. Файл ресурсов может содержать следующие ресурсы:

- значки;
- растровые изображения;
- курсоры.

Перечисленные компоненты являются ресурсами Windows, поскольку они разработаны и интерпретируются в соответствии со стандартами этой операционной системы.

Первоначально файл ресурсов содержит значок проекта, которым по умолчанию является изображение факела. В дальнейшем его можно изменить или заменить.

Для работы с файлами ресурсов в состав C++ Builder включен графический редактор Image Editor версии 3.0, вызываемый командой Tools\Image Editor (Средства\Редактор изображений).

Файл ресурсов имеет иерархическую структуру, в которой ресурсы разбиты на группы, и каждый ресурс имеет уникальное в пределах группы имя. Имя ресурса задается при его создании и в последующем используется в приложении для доступа к этому ресурсу. Значок проекта находится в группе Icon и по умолчанию имеет имя MAINICON.

Кроме файла с расширением res, объединяющего несколько ресурсов, редактор Image Editor также позволяет работать с файлами, содержащими следующие ресурсы (в скобках указано расширение имени файла):

- значки компонентов (dcr);
- растревые изображения (bmp);
- значки приложений (ico);
- курсоры (cur).

Параметры проекта

Для установки параметров проекта используется окно параметров проекта (Project Options), открываемое командой Project\Options (Проект\Параметры) или нажатием комбинации клавиш <Shift>+<Ctrl>+<F11>. Параметры разбиты на группы, каждая из которых располагается в окне параметров проекта на своей странице (рис. 11.6).

После установки отдельных параметров C++ Builder автоматически вносит нужные изменения в соответствующие файлы проекта. Так, параметры из страниц Forms и Application вносятся в файлы проекта и ресурсов, а параметры из страниц Compiler и Linker — в файл параметров проекта.

Ниже для примера приводятся фрагменты файла параметров проекта.

<IDEOPTIONS>

[Version Info]

```

IncludeVerInfo=0
...
DLL=0
Locale=1049
CodePage=1251
...
[Debugging]
DebugSourceDirs=$(BCB)\source\vcl

[Parameters]
RunParams=
Launcher=
UseLauncher=0
...
[Compiler]
ShowInfoMsgs=0
LinkDebugVcl=0
LinkCGLIB=0

```

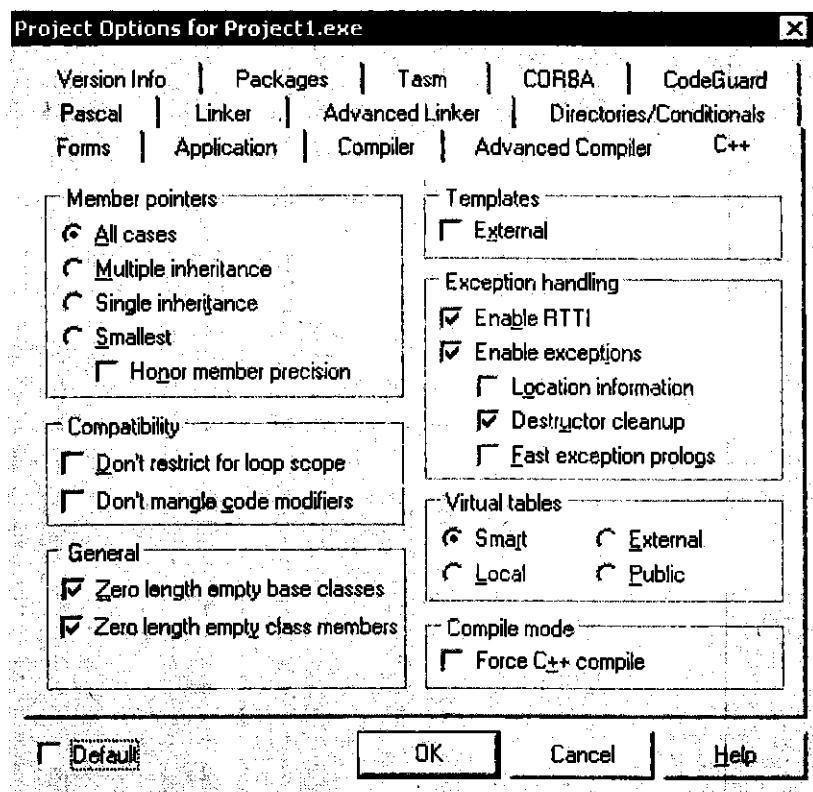


Рис. 11.6. Окно параметров проекта

Как видим, файл параметров проекта представляет собой текстовый файл, в котором построчно записаны параметры и их значения.

11.3. Компиляция и выполнение проекта

В процессе компиляции проекта создается готовый к использованию файл, которым может быть приложение (расширение exe) или динамически загружаемая библиотека (расширение dll). Как говорилось выше, мы будем рассматривать только приложения. Имя приложения, получаемого в результате компиляции, совпадает с именем файла проекта, а само приложение является автономным и не требует для своей работы дополнительных файлов C++ Builder.

Замечания.

Если в процессе выполнения приложения динамически используются другие файлы, например, изображения или файлы справки, то эти файлы должны быть в наличии.

При создании приложений, работающих с базами данных, необходимы файлы, составляющие базу данных, а также процессор баз данных.

Компиляция выполняется вызовом команды Project\Compile <Project1> (Проект\Компилировать <Проект1>) или нажатием комбинации клавиш <Ctrl>+<F9>.

В команде содержится имя проекта, разработка которого осуществляется в настоящий момент (первоначально Project1). При сохранении проекта под другим именем соответственно должно быть изменено и имя проекта в команде меню.

Скомпилировать проект для получения приложения можно на любой стадии разработки проекта. Это удобно для проверки вида и правильности функционирования отдельных компонентов формы, а также для тестирования фрагментов создаваемого кода. При компиляции проекта выполняются действия, приведенные ниже.

- Компилируются файлы всех модулей, содержимое которых изменилось со времени последней компиляции. Если исходный текст модуля по каким-либо причинам недоступен компилятору, то он не перекомпилируется.
- Если в модуль были внесены изменения, то перекомпилируется не только этот модуль, но и модули, использующие его с помощью директивы препроцессора #include.
- Перекомпиляция модуля происходит также при изменениях объектного файла (расширение obj) или подключаемого файла, используемых данным модулем.

- После компиляции всех модулей проекта компилируется файл проекта и создается исполняемый файл приложения с именем файла проекта.

Помимо компиляции, может быть выполнена также *сборка* проекта. При сборке компилируются все файлы, входящие в проект, независимо от того, были в них внесены изменения или нет. Для сборки проекта предназначена команда **Project\Build <Project1>** (**Проект\Собрать <Проект1>**).

Запустить проект на выполнение можно как в среде C++ Builder, так и в среде Windows.

Выполнение проекта в среде C++ Builder осуществляется командой **Run\Run** или нажатием клавиши <F9>. При этом созданное приложение начинает свою работу. Если в файлы проекта вносились изменения, то предварительно выполняется компиляция проекта.

С помощью команды **Run\Make** задается компиляция и сборка проекта без немедленного выполнения приложения (полученного файла с расширением .exe). По существу именно последним (отсутствием запуска приложения) и отличается действие команды **Make** от команды **Run**.

Запущенное приложение работает так же, как и запущенное вне среды C++ Builder, однако имеются некоторые особенности:

- нельзя запустить вторую копию приложения;
- продолжить разработку проекта можно только после завершения работы приложения;
- при зацикливании (зависании) приложения его завершение необходимо выполнять средствами C++ Builder с помощью команды **Run\Program Reset** (**Выполнение\Перезапуск программы**) или комбинации клавиш <Ctrl>+<F2>.

Для отладки приложений в среде C++ Builder можно использовать средства отладчика. В среде Windows созданное приложение можно запустить, как любое другое приложенис, например, с помощью Проводника.

11.4. Разработка приложения

C++ Builder относится к системам визуального программирования, называемым также *системами RAD* (Rapid Application Development, быстрая разработка приложений). Разработка приложения в C++ Builder включает два взаимосвязанных этапа:

- создание пользовательского интерфейса приложения;
- определение функциональности приложения.

Пользовательский интерфейс приложения определяет способ взаимодействия пользователя и приложения, т. е. внешний вид формы (форм) при выполнении приложения и то, каким образом пользователь управляет приложением. Интер-

фейс конструируется путем размещения в форме компонентов, называемых *интерфейсными компонентами* или *элементами управления*. Создается пользовательский интерфейс приложения с помощью окна **Формы**, которое в среде разработки представляет собой модель формы времени выполнения.

Функциональность приложения определяется процедурами, которые выполняются при возникновении определенных событий, например, происходящих при действиях пользователя с элементами управления формы.

Таким образом, в процессе разработки приложения в форму помещаются компоненты, для них устанавливаются необходимые свойства и создаются обработчики событий.

Пример простейшего приложения

Создадим для примера простейшее приложение. Слово «создадим» в данном случае является не совсем точным, т. к. создавать и тем более программировать не придется вообще ничего: C++ Builder изначально предоставляет готовое приложение, состоящее из одной формы.

Сразу же после создания нового приложения C++ Builder предлагает разработчику «пустую» форму. Данная форма не является пустой в буквальном смысле слова — она содержит основные элементы окна Windows: заголовок **Form1**, кнопки сворачивания, разворачивания и закрытия окна, изменения размеров окна и кнопку вызова системного меню окна. Именно эта форма отображается при первом запуске C++ Builder в окне формы.

Любое приложение Windows выполняется в соответствующем окне. Даже если оно ничего не делает в смысле функциональности, т. е. является пустым, то все равно должно иметь свое окно. C++ Builder — это среда разработки приложений под Windows, поэтому для любого разрабатываемого приложения автоматически предлагается окно (форма), для которой уже созданы два файла — с описанием и модулем.

Итак, простейшее приложение создается автоматически каждый раз в начале работы над новым проектом и является отправной точкой для дальнейших действий. Это приложение имеет минимум того, что нужно любому приложению, выполняемому в среде Windows, и ни одним элементом больше.

Простейшее приложение представляет из себя заготовку или каркас, обеспечивающий разработчика всем необходимым для каждого приложения вообще. Так, не нужно писать свой обработчик событий клавиатуры или драйвер мыши, а также создавать пакет процедур для работы с окнами. Более того, нет необходимости интегрировать драйвер мыши с пакетом для работы с окнами. Это все уже выполнено создателями C++ Builder, и каркас приложения представляет собой полностью завершенное и функционирующее приложение, которое никаких действий не производит.

Отметим, что окно (а вместе с ним и приложение) действительно ничего не делает с точки зрения пользователя — оно не предоставляет функциональ-

ности, специфичной для каждого приложения. Вместе с тем это пустое окно выполняет достаточно большую работу с точки зрения программиста. Например, оно ожидает действий пользователя, связанных с мышью и клавиатурой, и реагирует на изменение своего размера, перемещение, закрытие и некоторые другие команды.

В полной мере оценить эти возможности окна может только программист, который писал приложения под Windows традиционным способом, т. е. вручную и без IDE. Иэнутри Windows представляет собой систему с индексами, контекстами, обратными вызовами и множеством других сложнейших элементов, которые надо знать, которыми нужно управлять и в которых легко запутаться. Однако поскольку эти элементы имеются в каждом функционирующем приложении Windows, достаточно написать их один раз и в дальнейшем уже пользоваться готовыми блоками. Именно это и осуществляет система C++ Builder, избавляя тем самым программиста от сложной рутинной работы.

При компиляции проекта можно использовать специальные пакеты динамически загружаемых библиотек (DLL), что позволяет в значительной степени уменьшить размер приложения. Однако при этом приложение уже не является автономным и в процессе своей работы обращается к пакетам, которые были задействованы при компиляции проекта.

При конструировании приложения разработчик добавляет к простейшему приложению новые формы, управляющие элементы, а также новые обработчики событий.

Создание пользовательского интерфейса

Пользовательский интерфейс приложения составляют компоненты, которые разработчик выбирает в Палитре компонентов и размещает в форме. При конструировании интерфейса приложения действует принцип WYSIWYG (What You See Is What You Get — «что видите, то и получаете»), и разработчик при создании приложения видит форму почти такой же, как и при его выполнении.

Компоненты являются структурными единицами и делятся на визуальные (видимые) и невизуальные (системные). При этом понятия «видимый» и «невидимый» относятся только к этапу выполнения, на этапе проектирования видны все компоненты приложения.

К *визуальным компонентам* относятся, например, кнопки, списки или переключатели, а также собственно форма. Так как с помощью визуальных компонентов пользователь управляет приложением, их также называют *управляющими компонентами* или *элементами управления*. Именно визуальные компоненты образуют пользовательский интерфейс приложения.

К *невизуальным компонентам* относятся компоненты, выполняющие вспомогательные, но не менее важные действия, например, таймер Timer (позволяет отсчитывать интервалы времени).

При создании интерфейса приложения для каждого компонента выполняются следующие операции:

- выбор компонента в Палитре компонентов и размещение его в форме;
- изменение свойств компонента.

Разработчик выполняет эти операции в окне Формы, используя Палитру компонентов и Инспектор объектов. При этом действия разработчика похожи на работу в среде графического редактора, а сам процесс создания интерфейса приложения больше напоминает конструирование или рисование, чем традиционное программирование. В связи с этим часто работу по созданию интерфейса называют конструированием.

Выбор компонента в Палитре компонентов выполняется щелчком мыши на нужном компоненте, например на кнопке Button, в результате чего его значок принимает уточленный (нажатый) вид. Если после этого щелкнуть на свободном месте формы, то на ней появляется выбранный компонент, а его значок в Палитре компонентов принимает обычный (ненажатый) вид. Значки компонентов отражают назначение компонентов, и при наличии небольших практических навыков выбор нужного компонента происходит достаточно быстро. Кроме того, при наведении на каждый компонент указателя мыши отображается всплывающая подсказка с информацией о его назначении.

Обозначения классов (типов объектов) в C++ Builder, в том числе компонентов, начинаются с буквы T. Иногда типы (TButton, TLabel) используются вместо имен (Button, Label) для обозначения компонентов. Мы будем использовать для компонентов имена или типы в зависимости от ситуации.

После размещения компонента в форме система C++ Builder автоматически вносит изменения в файлы модуля и представления формы. В описание класса формы (заголовочный файл модуля формы) для каждого нового компонента добавляется строчка формата

<Тип компонента> *<Имя компонента>;

Имя нового компонента является значением его свойства Name, а тип совпадает с типом выбранного в Палитре компонента. Например, для кнопки Button1 эта строчка первоначально будет иметь вид:

TButton *Button1;

В файле представления для кнопки Button может быть записан следующий код:

```
object Button1: TButton
  Left = 98
  Top = 110
  Width = 75
  Height = 25
  Caption = 'Button1'
  TabOrder = 0
end
```

Для размещения в форме нескольких одинаковых компонентов удобно перед выбором компонента в Палитре компонентов нажать и удерживать клавишу <Shift>. В этом случае после щелчка мыши в области формы и размещения там выбранного компонента его значок в Палитре остается утопленным, и каждый последующий щелчок в форме приводит к появлению в ней еще одного такого же компонента. Для отмены выбора этого компонента достаточно выбрать другой компонент или щелкнуть мышью на изображении стрелки в левом углу Палитры компонентов.

После размещения компонента в форме можно с помощью мыши изменять его положение и размеры. В случае нескольких компонентов можно выполнять выравнивание или перевод того или иного компонента на передний или задний план. При этом действия разработчика не отличаются от действий в среде обычного графического редактора. Одновременно выделить в форме несколько компонентов можно щелчками мыши на них при нажатой клавише <Shift>.

По умолчанию компоненты выравниваются в форме по линиям сетки, что определяет флагок **Snap to grid** (Выравнивать по сетке), входящий в набор параметров интегрированной среды разработки. В ряде случаев этот флагок приходится отключать, например, при плотном размещении компонентов в форме. По умолчанию шаг сетки равен восьми пикселям, а сетка при проектировании отображается на поверхности формы. Необходимость выравнивания по сетке, видимость сетки (флагок **Display grid** (Отображать сетку)) и размер шага сетки по горизонтали и вертикали устанавливаются на вкладке **Designer** (Конструктор) диалогового окна **Environment Options** (Параметры среды), вызываемого одноименной командой меню **Tools** (Средства).

Внешний вид компонента определяется его свойствами, которые становятся доступными в окне Испектора объектов, когда компонент выделен в форме и вокруг него появились маркеры выделения (рис. 11.7). Доступ к свойствам самой формы осуществляется аналогично, однако в выбранном состоянии форма не выделяется маркерами. Для выделения (выбора) формы достаточно щелкнуть в любом ее месте, свободном от других компонентов.

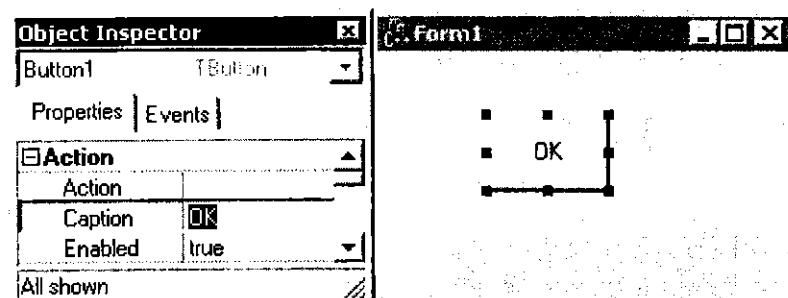


Рис. 11.7. Доступ к свойствам компонента

В раскрывающемся списке, расположенному в верхней части окна Инспектора объектов, отображаются имя компонента и его тип. Выбрать тот или иной компонент и, соответственно, получить доступ к его свойствам можно в этом списке Инспектора объектов. Такой способ выбора удобен в случаях, когда компонент полностью закрыт другими объектами.

В нижней части окна Инспектора объектов слева приводятся имена всех свойств компонента, которые доступны на этапе разработки приложения. Справа для каждого свойства стоит его значение. Отметим, что кроме этих свойств компонент может иметь и свойства, которые доступны только *во время выполнения* приложения.

Свойства представляют собой атрибуты, определяющие способ отображения и функционирования компонентов при выполнении приложения. Первоначально значения свойств задаются по умолчанию. После помещения компонента в форму его свойства можно изменять в процессе проектирования или в ходе выполнения приложения.

Управление свойствами в процессе проектирования заключается в изменении значений свойств компонентов непосредственно в окне формы («рисование») или с помощью Инспектора объектов.

Разработчик может изменить значение свойства компонента, введя или выбрав нужное значение. При этом одновременно изменяется соответствующий компонент, т. е. уже при проектировании видны результаты сделанных изменений. Например, при изменении заголовка кнопки (свойство **Caption**) оно сразу же отображается на ее поверхности.

Для подтверждения нового значения свойства достаточно нажать клавишу <Enter> или перейти к другому свойству или компоненту. Отмена изменений производится клавишей <Esc>. Если введено недопустимое для свойства значение, то выдается предупреждающее сообщение, а изменение значения отвергается. Изменения свойств автоматически учитываются в файле описания формы, используемом компилятором при создании формы, а при изменении свойства **Name** изменения вносятся и в описание класса формы.

Для большинства свойств компонентов (например, для свойств **Color** (цвет), **Caption** (заголовок) и **Visible** (видимость)) имеются значения по умолчанию.

Для обращения к компоненту в приложении предназначено свойство **Name**, первоначальное значение которого образуется автоматически следующим образом: к имени компонента добавляется его номер в порядке помещения в форму. Например, первая кнопка **Button** получает имя **Button1**, вторая — **Button2** и т. д. Первоначально от значения свойства **Name** получает свое значение и свойство **Caption**.

Обычно разработчик предпочитает дать компонентам более информативные имена, чем имена по умолчанию. При этом целесообразно включать в имя данные о типе компонента и его назначении в приложении. Так, кнопке типа **TButton**, предназначеннной для закрытия окна, можно присвоить имя **btnClose**

или ButtonClose. Каждый разработчик самостоятельно устанавливает удобные правила именования компонентов. Для простоты мы будем часто использовать имена, назначаемые по умолчанию, например, Form1, Button1 или Edit1.

Значения свойств, связанных с размерами и положением компонента (например, Left и Top), автоматически изменяются при перемещении компонента мышью и настройке его размеров.

Если в форме выделено несколько компонентов, то в окне Инспектора объектов доступны свойства, общие для всех этих компонентов. При этом сделанные в окне Инспектора объектов изменения действуют на все выделенные компоненты.

Для установки значений свойств в Инспекторе объектов используются подключающиеся автоматически редакторы свойств:

- **простой** (текстовый) – значение свойства вводится или редактируется как обычная строка символов, которая интерпретируется как числовой или строковый тип C++ Builder; используется для таких свойств, как Caption, Left, Height и Hint;
- **перечисляемый** – значение свойства выбирается в раскрывающемся списке. Список раскрывается щелчком на стрелке, которая появляется при установке указателя мыши в области значения свойства. Можно не выбирать нужное значение, а ввести его с клавиатуры, однако на практике это обычно не делается, т. к. допускаются только предлагаемые значения. Кроме того, возрастает трудоемкость и увеличивается вероятность ошибки. Используется для таких свойств, как FormStyle, Visible и ModalResult;
- **множественный** – значение свойства представляет собой комбинацию значений из предлагаемого множества. В окне Инспектора объектов слева от имени свойства множественного типа стоит знак «+». Формирование значения свойства выполняется с помощью дополнительного списка, раскрываемого двойным щелчком на имени свойства. Этот список содержит перечень всех допустимых значений свойства, справа от каждого значения можно указать true или false. Выбор true означает, что данное значение включается в комбинацию значений, а false – нет. Используется для таких свойств, как BorderIcons и Anchors;
- **объекта** – свойство является объектом и, в свою очередь, содержит другие свойства (подсвойства), каждое из которых можно редактировать отдельно. Используется для таких свойств, как Font, Items и Lines. В области значения свойства-объекта в скобках указывается тип объекта, например, (TFont) или (TSrings). Для свойства-объекта слева от имени может стоять знак «+», в этом случае управление его свойствами выполняется так же, как и для свойства множественного типа, т. е. через раскрывающийся список. Этот список в левой части содержит имена подсвойств, а в правой – значения, редактируемые обычным

способом. В области значения может отображаться кнопка с тремя точками. Это означает, что для данного свойства имеется специальный редактор, который вызывается нажатием этой кнопки. Так, для свойства **Font** открывается стандартное окно Windows для установки параметров шрифта.

При выполнении приложения значения свойств компонентов (доступных в окне Инспектора объектов) можно изменять с помощью инструкций присваивания, например, в обработчике события создания формы. Так, изменение заголовка кнопки **Button1** можно выполнить следующим образом:

```
Button1->Caption = "OK";
```

Такой способ требует, однако, большего объема работ, чем в случае использования Инспектора объектов, кроме того, подобные установки вступают в силу только во время выполнения приложения и на этапе разработки не видны, что в ряде случаев затрудняет управление визуальными компонентами. Тем не менее, для наглядности во многих примерах значения отдельных свойств мы будем устанавливать с помощью операторов присваивания, а не через Инспектор объектов.

Отметим, что есть свойства времени выполнения, недоступные через Инспектор объектов, с которыми можно работать только во время выполнения приложения. К таким свойствам относятся, например, число записей **RecordCount** набора данных или поверхность рисования **Canvas** визуального компонента.

Определение функциональности приложения

На любой стадии разработки интерфейсной части приложение можно запустить на выполнение. После компиляции на экране появляется форма приложения, которая выглядит примерно так же, как она была сконструирована на этапе разработки. Форму можно перемещать по экрану, изменять ее размеры, сворачивать и разворачивать, а также закрывать нажатием соответствующей кнопки в заголовке или другим способом. То есть форма ведет себя, как обычное окно Windows.

Реакция на приведенные действия присуща каждой форме и не зависит от назначения приложения и его особенностей. В форме, как правило, размещены компоненты, образующие интерфейс приложения, и задача разработчика — определить для этих компонентов нужную реакцию на те или иные действия пользователя, например, на нажатие кнопки или выбор переключателя. Эта реакция и определяет *функциональность* приложения.

Допустим, что при создании интерфейса приложения в форме была размещена кнопка **Button1**, предназначенная для закрытия окна. По умолчанию эта кнопка получила имя и заголовок **Button1**, однако заголовок (свойство **Caption**) с помощью окна Инспектора объектов было заменено более осмысленным — ОК. При выполнении приложения кнопку **Button1** мож-

но нажимать с помощью мыши или клавиатуры. Кнопка отображает нажатие визуально, однако никаких действий, связанных с закрытием формы, не выполняется. Подобное происходит потому, что для кнопки не определена реакция на какие-либо действия пользователя, в том числе и на ее нажатие.

Чтобы кнопка могла реагировать на то или иное событие, для него необходимо создать или указать процедуру обработки, которая будет вызываться при возникновении этого события. Для создания процедуры обработки события, или обработчика, нужно прежде всего выделить в форме кнопку и перейти на страницу **Events** (События) окна Инспектора объектов, где указываются все возможные для данной кнопки события (рис. 11.8).

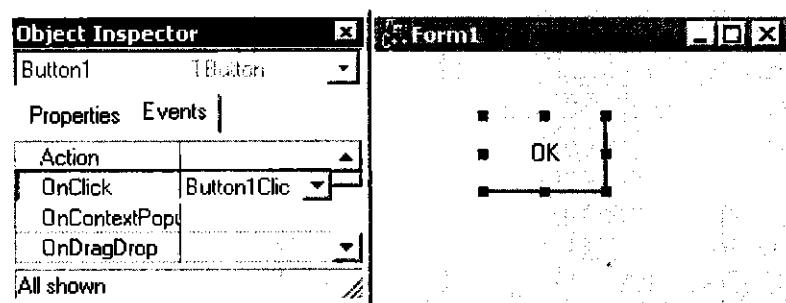


Рис. 11.8. Доступ к событиям компонента

Так как при нажатии кнопки возникает событие **OnClick**, следует создать обработчик именно этого события. Нужно сделать двойной щелчок в области значения события **OnClick**, в результате C++ Builder автоматически создает в модуле формы заготовку процедуры-обработчика. При этом окно Редактора кода переводится на передний план, а курсор устанавливается в то место процедуры, где программист должен написать код, выполняемый при нажатии кнопки **Button1**. Поскольку при нажатии кнопки **OK** должно закрываться окно, то в этом месте можно указать **Form1->Close()** или просто **Close()**. Файлы в составе модуля формы будут иметь следующий вид:

Заголовочный файл:

```
//-----
#ifndef Unit1H
#define Unit1H
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
```

```
#include <Forms.hpp>
//-----
class TForm1 : public TForm
{
public: // IDE-managed Components
    // Объявление объекта кнопки типа TButton с именем OK
    TButton *OK;
    // Заголовок процедуры обработчика
    void __fastcall OKClick(TObject *Sender);
private: // User declarations
public: // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif
Файл реализации:
//-----
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TForm1::OKClick(TObject *Sender)
{
Form1->Close();
}
//-----
```

Здесь полужирным начертанием выделен код, набранный программистом; все остальное среда C++ Builder создала автоматически, в том числе и включение заголовка процедуры-обработчика в описание класса формы Form1.

В обработчике событий (приведенном в примере и в других обработчиках) параметр **Sender** указывает на компонент, который получил событие и

вызвал обработчик. Иногда полезно иметь общий обработчик для разных объектов, который ведет себя по-разному в зависимости от класса вызвавшего его объекта. В такой ситуации использование параметра **Sender** может быть очень полезным. Заметим, что в заголовках функции он объявлен как указатель на объект типа **TObject** в виде **TObject *Sender**.

Класс **TObject** является базовым классом для всех классов в C++ Builder, но в нем нет свойств, пригодных к использованию в обработчиках событий. В этой связи при необходимости обращения к некоторым свойствам объектов нужно выполнять приведение типа параметра **Sender** к классу, в котором нужные свойства объявлены.

К примеру, пусть в обработчике требуется задать команду, помещающую в односторонний редактор **Edit** текст **Выполнен щелчок на компоненте <имя компонента>**. Имя компонента определяется значением его свойства **Name**. Поскольку в классе **TObject** это свойство отсутствует, тип параметра нужно приводить к классу **TComponent**, в котором это свойство появляется впервые, или к любому его наследнику, например **TControl**. В результате решающий поставленную задачу оператор можно записать в виде:

```
Edit1->Text = "Выполнен щелчок на компоненте "+((TComponent *)Sender)->Name;
```

или

```
Edit1->Text = "Выполнен щелчок на компоненте "+((TControl *)Sender)->Name;
```

Здесь выражения **((TComponent *)Sender)** и **((TControl *)Sender)** задают приведение типа параметра **Sender** к типу указателя на объект соответствующего класса.

Среда C++ Builder обеспечивает автоматизацию набора кода при вызове свойств и методов объектов и записи стандартных конструкций языка C++. Так, после указания имени объекта и разделяющего знака **->** автоматически появляется список (рис. 11.9), содержащий доступные свойства и методы этого объекта. При необходимости с помощью комбинации клавиш **<Ctrl>+<пробел>** можно обеспечить принудительный вызов этого списка. Имя выбранного свойства или метода автоматически добавляется справа от знака **->**. Если метод содержит параметры, то отображается подсказка, содержащая состав и типы параметров.

Перечень стандартных конструкций языка вызывается нажатием комбинации клавиш **<Ctrl>+<J>**. После выбора требуемой конструкции автоматически добавляется ее код. Например, при выборе оператора **for** в коде появится следующий текст:

```
for (;;) {
```

Имя обработчика **TForm1::OKClick** образуется прибавлением к имени компонента имени события без префикса **On**. Это имя является значением события, для которого создан обработчик, в нашем случае – для события **OnClick**.

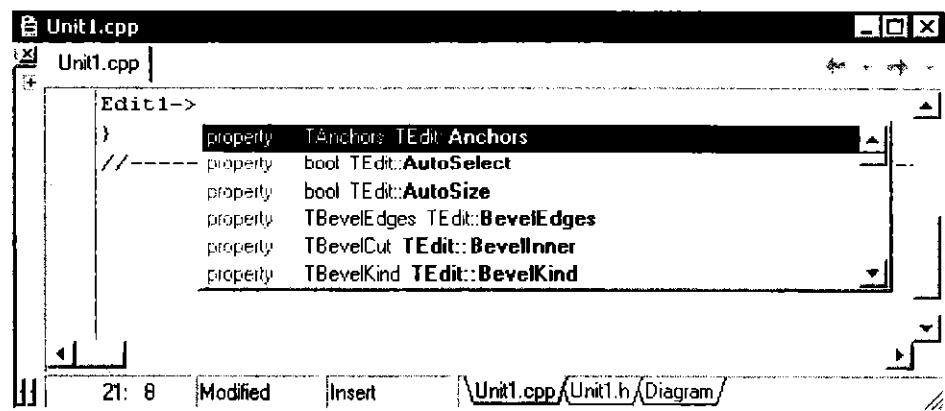


Рис. 11.9. Список свойств и методов объекта Edit1

нажатия кнопки с именем ОК. При изменении через окно Инспектора объектов имени кнопки происходит автоматически переименование этой процедуры во всех файлах (.dfm и .cpp) проекта.

Аналогично создаются обработчики для других событий и других компонентов. Более подробно события рассматриваются при изучении соответствующих компонентов.

Для удаления процедуры-обработчика достаточно удалить код, который программист внес в нее самостоятельно. После этого при сохранении или компиляции модуля обработчик будет удален автоматически из всех файлов проекта.

При удалении какого-либо компонента все его пустые обработчики остаются в модуле формы.

Вместо создания нового обработчика для события можно выбрать существующий обработчик, если такой имеется. Для этого нужно в окне Инспектора объектов щелчком на стрелке в области значения свойства раскрыть список процедур, которые можно использовать для обработки этого события. События объекта тоже являются свойствами и имеют определенный для них тип. Для каждого события можно назначить обработчик, принадлежащий к типу этого события. После выбора в списке нужной процедуры она назначается обработчиком события.

Одну и ту же процедуру можно связать с несколькими событиями, в том числе для различных компонентов. Такая процедура называется *общим (разделяемым) обработчиком* и вызывается при возникновении любого из связанных с ней событий. В теле общего обработчика можно предусмотреть действия, позволяющие определить, для какого именно компонента или события вызвана процедура, и в зависимости от этого выполнить нужные команды.

11.5. Средства интегрированной среды разработки

Интегрированная среда разработки имеет в своем составе много различных средств, служащих для удобной и эффективной разработки приложений. В этом разделе мы оценим наиболее обиные элементы интегрированной среды разработки C++ Builder.

Управление параметрами среды

Пользователь может управлять интегрированной средой разработки, настраивая ее отдельные параметры, например, появление окна, отображающего ход компиляции проекта, или автоматическое сохранение редактируемых файлов. Установка параметров выполняется в диалоговом окне **Environment Options** (Параметры среды), вызываемом командой **Tools\Environment Options** (Средства\Параметры среды). Все параметры объединены по группам, размещенным на отдельных страницах (рис. 11.10).

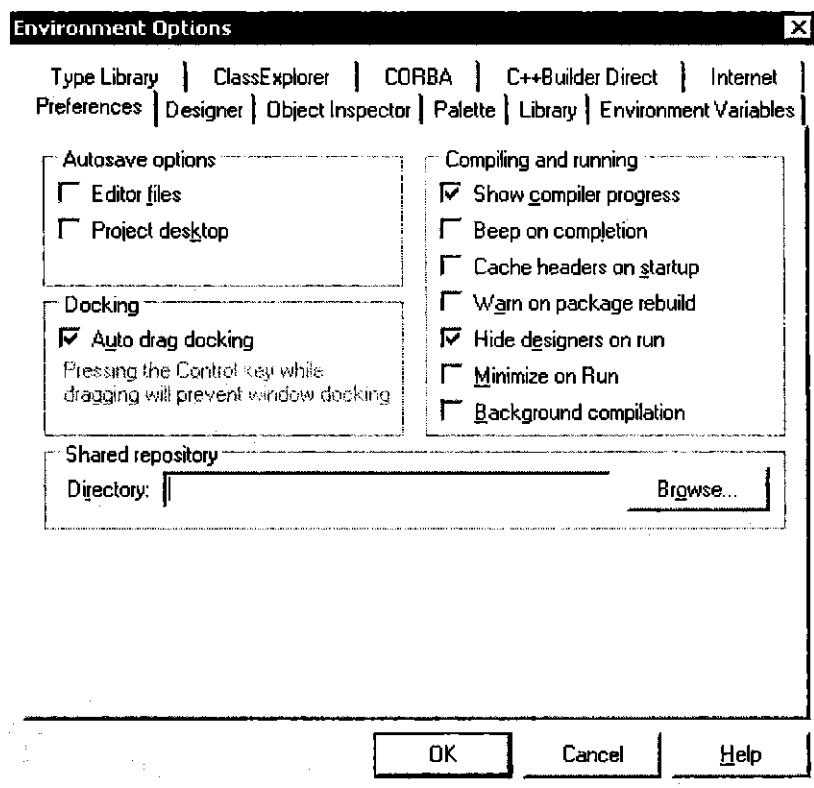


Рис. 11.10. Окно настройки параметров среды разработки

Параметры среды C++ Builder для каждого проекта сохраняются в файле конфигурации (project configuration file) с расширением cfg.

Встроенный отладчик

Интегрированная среда разработки включает встроенный отладчик приложений, в значительной степени облегчающий поиск и устранение ошибок в приложениях. Средства отладчика доступны через команды меню Run и подменю View\Debug Windows (Просмотр\Окна отладки) и позволяют работать в следующих режимах:

- выполнение до указанной строки кода;
- пошаговое выполнение приложения;
- выполнение до точки останова (Breakpoint);
- включение и выключение точек останова;
- просмотр значений объектов, например, переменных, в окне просмотра;
- установка значений объектов при выполнении приложения.

Установка параметров отладчика выполняется в диалоговом окне Debugger Options (Параметры отладчика), вызываемом одноименной командой меню Tools (рис. 11.11).

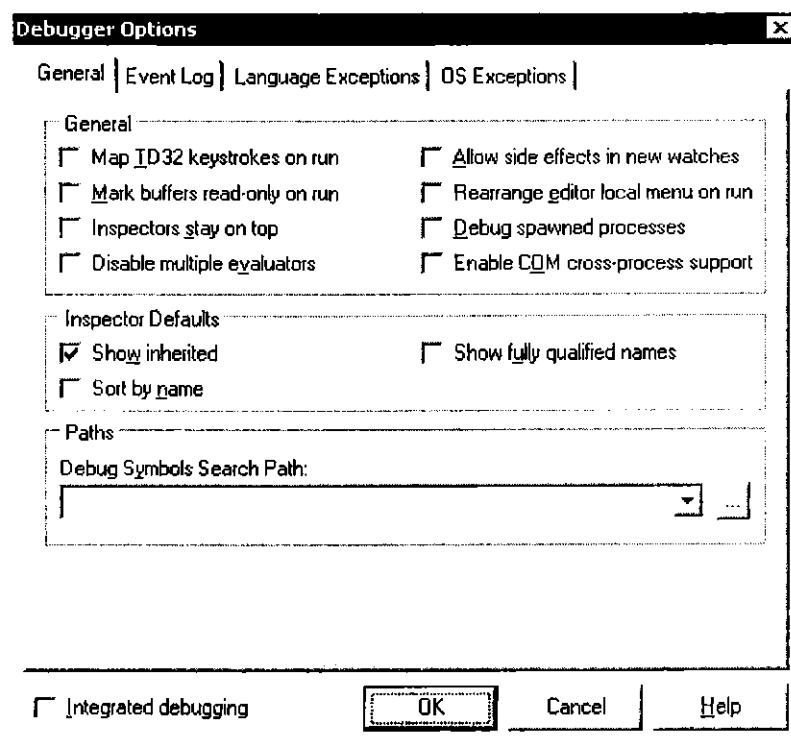


Рис. 11.11. Окно установки параметров отладчика

Включением/выключением отладчика управляет флажок **Integrated debugging** (**Интегрированная отладка**), который по умолчанию установлен, и отладчик автоматически подключается к каждому приложению. В ряде случаев, например при отладке обработчиков исключений и проверке собственных средств обработки ошибок, этот флажок целесообразно снять.

Хранилище объектов

Система C++ Builder позволяет многократно использовать одни и те же объекты в качестве шаблонов для дальнейшей разработки приложений. Для хранения таких объектов используется специальное Хранилище объектов, или Репозиторий (**Repository**).

Вставить в приложение новый объект можно, открыв командой **File\New\Other** (**Файл\Новый\Другой**) окно **New Items** (**Новые элементы**) для выбора нового объекта в хранилище. Это окно можно также открыть нажатием кнопки **New** панели инструментов Менеджера проектов.

В Хранилище находятся различные объекты, например, шаблоны приложений, форм, отчетов, а также Мастера форм. Все объекты объединены в группы, размещенные на отдельных страницах, например:

- **New** – встроенные базовые объекты, используемые при разработке приложений;
- **ActiveX** – объекты COM и OLE, элементы ActiveX, библиотеки ActiveX, активные серверные страницы (ASP);
- **Project1** – формы создаваемого приложения;
- **Forms** – формы;
- **Dialogs** – диалоговые окна (стандартное, справочное, для ввода пароля);
- **Projects** – проекты одно- и многодокументного приложений;
- **Data Modules** – модули данных;
- **Web Documents** – Web-документы (HTML, XHTML, WML, XSL).

Название страницы **Project1** совпадает с названием создаваемого проекта, а сама страница содержит в качестве шаблонов уже созданные формы приложения (первоначально это одна форма с именем **Form1**). При изменении названия проекта или формы соответственно изменяются их названия в Хранилище объектов. При добавлении к проекту новой формы ее шаблон автоматически добавляется на страницу проекта. В случае удаления из проекта формы ее шаблон также автоматически исключается из Хранилища объектов.

Для добавления нового объекта к проекту необходимо перейти на нужную страницу и указать объект. При нажатии кнопки **OK** происходит добавление объекта. Объекты можно добавлять к проекту различными способами, зависящими от выбранного переключателя в нижней части окна выбора нового объекта.

- **Copy** — в проект добавляется копия объекта из Хранилища. В проекте этот объект можно изменять, однако все изменения являются *локальными* в пределах проекта и не затрагивают оригинал, находящийся в Хранилище объектов.
- **Inherit** — от объекта из Хранилища порождается (наследуется) новый объект, который и добавляется к проекту. Разработчик может добавлять к объекту новые компоненты, а также изменять свойства уже существующих элементов, не связанные с их именами. При модификации этого объекта в проекте невозможно удалить какую-либо его составную часть (компонент) или изменить имя (свойство *Name*). По умолчанию подобным образом к проекту добавляются объекты (обычно формы) созданного проекта, расположенные на странице *Project1*.
- **Use** — в проект включается непосредственно сам объект из Хранилища со всеми своими файлами. При изменении в проекте этого объекта изменяется и объект в Хранилище, а также объекты в других проектах, которые таким же образом используют этот объект.

Настройка состава объектов в Хранилище объектов при необходимости выполняется в окне *Object Repository*, открываемом командой *Tools\Repository*.

В процессе настройки в Хранилище объектов можно добавлять (кнопка *Add Page*), удалять из него (кнопка *Delete Page*) и переименовывать страницы (кнопка *Rename Page*), а также редактировать (кнопка *Edit Object*) и удалять (кнопка *Delete Object*) объекты.

Объекты приложения, формы, фрейма, модуля данных и модуля кода тоже можно добавить к проекту через подменю *File\New*, в котором содержатся команды добавления к проекту объектов *Application*, *CLX Application*, *Data Module*, *Form*, *Frame* и *Unit*.

11.6. Базы данных и средства работы с ними

Далее рассматриваются основные элементы, составляющие реляционную базу данных (таблицы, ключи и индексы, связи между таблицами) и форматы таблиц, используемые в системе C++ Builder. Указываются инструментальные средства системы C++ Builder, используемые при работе с базами данных, а также компоненты, используемые при разработке приложений для баз данных. Кроме того, освещаются особенности организации обработки исключений при работе с базами данных.

Характеристика механизмов доступа к данным

Одно- и двухуровневые приложения C++ Builder могут осуществлять доступ к локальным и удаленным БД с использованием следующих механизмов:

- **BDE** (Borland Database Engine – процессор баз данных фирмы Borland), предоставляющий развитый интерфейс API для взаимодействия с базами данных (представляет собой набор динамических библиотек и драйверов, предназначенных для организации доступа к БД);
- **ADO** (ActiveX Data Objects – объекты данных ActiveX) осуществляет доступ к информации с помощью OLE DB (Object Linking and Embedding Data Base – связывание и внедрение объектов баз данных);
- **dbExpress** обеспечивает быстрый доступ к информации в базах данных с помощью набора драйверов;
- **InterBase** реализует непосредственный доступ к базам данных InterBase.

Выбор варианта технологии доступа к информации в базах данных, кроме прочих соображений, определяется с учетом удобства подготовки разработанного приложения к распространению, а также дополнительного расхода ресурсов памяти. К примеру, инсталляция для BDE требует примерно 15 Мбайт внешней памяти на диске и настройки псевдонимов используемых баз данных. Вариант InterBase вряд ли можно назвать конкурентоспособным, поскольку он ориентирован строго на работу с одноименным сервером баз данных.

Трехуровневые приложения C++ Builder можно создавать с помощью механизма DataSnap. Используемые при создании трехуровневых (многоуровневых) приложений баз данных компоненты расположены на страницах DataSnap и Data Access Палитры компонентов.

BDE представляет собой совокупность динамических библиотек и драйверов, обеспечивающих доступ к данным. Процессор BDE должен устанавливаться на всех компьютерах, на которых выполняются C++ Builder-приложения, осуществляющие работу с БД. Приложение через BDE передает запрос к базе данных, а обратно получает требуемые данные. Механизм BDE до последней версии системы C++ Builder получил самое широкое распространение ввиду широкого спектра предоставляемых им возможностей. Идеологи фирмы Borland планируют отказаться от его поддержки, заменив его механизмом dbExpress. Мы приводим множество примеров и описание технологии применения BDE для работы с базами данных в связи с тем, что накоплено большое количество приложений с использованием этого подхода.

Механизм **ADO** доступа к информации базы данных является стандартом фирмы Microsoft. Использование этой технологии подразумевает использование настраиваемых провайдеров данных. Технология ADO обеспечивает универсальный механизм доступа из приложений к информации источников данных. Эта технология основана на стандартных интерфейсах COM, являющихся системным механизмом Windows. Это позволяет удобно распространять приложения баз данных без вспомогательных библиотек.

Механизм доступа **dbExpress** подразумевает использование совокупности драйверов, компонентов, инкапсулирующих соединения, транзакций, запросов, наборов данных и интерфейсов. С ее помощью обеспечивается универсальный доступ к функциям этого механизма. Обеспечение взаимодействия с серверами баз данных по технологии dbExpress основано на использовании специализированных драйверов. Последние для получения данных применяют запросы SQL. На стороне клиента при этом нет кэширования данных, здесь применяются только односторонние курсоры и не обеспечивается возможность прямого редактирования наборов данных.

Далее рассматривается работа с локальными БД с помощью механизма BDE. Соответствующая локальная архитектура информационной системы приведена на рис. 11.12. Работа с БД происходит, как правило, в *однопользовательском* режиме. При необходимости можно запустить на компьютере другое приложение, одновременно осуществляющее доступ к этим же данным. Для управления совместным доступом к БД необходимы специальные средства контроля и защиты. Эти средства могут понадобиться, например, в случае, когда приложение пытается изменить запись, которую редактирует другое приложение. Каждая разновидность БД осуществляет подобный контроль своими способами и обычно имеет встроенные средства разграничения доступа.

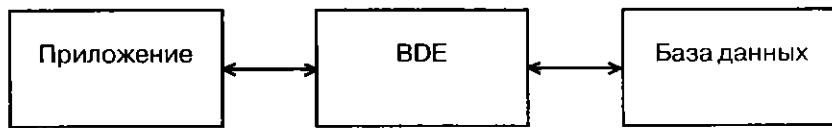


Рис. 11.12. Локальная архитектура с BDE

Для доступа к локальной БД процессор баз данных BDE использует стандартные драйверы, которые позволяют работать с форматами БД dBase, Paradox, FoxPro, а также с текстовыми файлами.

Таблицы баз данных

Напомним, что реляционная база данных (БД) состоит из взаимосвязанных таблиц. Каждая таблица содержит информацию об объектах одного типа, а совокупность всех таблиц образует единую БД.

Таблицы, образующие БД, находятся в каталоге (папке) на жестком диске. Таблицы хранятся в файлах и похожи на отдельные документы или электронные таблицы (например, табличного процессора Microsoft Excel), их мож-

но перемещать и копировать обычным способом, скажем, с помощью Проводника Windows. Однако в отличие от документов, таблицы БД поддерживают **многопользовательский** режим доступа, это означает, что их могут одновременно использовать несколько приложений.

Для одной таблицы создается несколько файлов, содержащих данные, индексы, ключи и т. п. Главным из них является файл с данными, имя этого файла совпадает с именем таблицы, которое задается при ее создании. В некотором смысле понятия таблицы и ее главного файла являются синонимами, при выборе таблицы выбирается именно ее главный файл: для таблицы dBase это файл с расширением dbf, а для таблицы Paradox – файл с расширением db. Имена остальных файлов таблицы назначаются автоматически – все файлы имеют одинаковые имена, совпадающие с именами таблиц, и разные расширения, указывающие на содержимое соответствующего файла. Расширения файлов приведены ниже в данной главе в разделе «*Форматы таблиц*».

Поле таблицы содержит данные одного из допустимых типов, например, строкового, целочисленного или типа «дата». При вводе значения в поле таблицы автоматически производится проверка соответствия типа значения и типа поля. В случае, когда эти типы не совпадают, а преобразование типа значения невозможно, генерируется исключение.

Особенности организации таблиц зависят от конкретной СУБД, используемой для создания и ведения БД. Их следует учитывать при выборе типа (формата) таблицы, т. к. они влияют не только на организацию БД, но и на построение приложения для работы с этой БД. Однако, несмотря на все различия таблиц, существуют общие правила создания и ведения БД, а также разработки приложений, которые и будут далее рассмотрены.

Основу таблицы составляет описание ее полей, каждая таблица должна иметь хотя бы одно поле. Как отмечалось, понятие структуры таблицы является более широким и включает: описание полей, ключ, индексы, ограничения на значения полей, ограничения ссылочной целостности между таблицами, пароли.

Отметим, что отдельные элементы структуры зависят от формата таблиц, например, для таблиц dBase нельзя задать ограничения ссылочной целостности (т. к. у них нет ключей). Все элементы структуры задаются на физическом уровне (уровне таблицы) и действуют для всех программ, выполняющих операции с БД, включая средства разработки и ведения БД (например, программу Database Desktop). Многие из этих элементов (например, ограничения на значения полей или поля просмотра) можно также реализовать в приложении программно, однако в этом случае они действуют только в пределах своего приложения.

Как отмечалось, с таблицей в целом можно выполнять следующие операции: создание, изменение структуры, переименование, удаление.

При *создании* таблицы задаются структура и имя таблицы. При сохране-

нии на диске создаются все необходимые файлы, относящиеся к таблице. Их имена совпадают с именем таблицы.

При *изменении* структуры таблицы в ней могут измениться имена и характеристики полей, состав и наименования ключа и индексов, ограничения. Однако имена таблицы и ее файлов остаются прежними.

При *переименовании* таблица получает новое имя, в результате чего новое имя также получают все ее файлы. Для этого используются соответствующие программы (утилиты), предназначенные для работы с таблицами БД, например, Database Desktop или Data Pump. Отметим, что таблицу нельзя переименовать, просто изменив названия всех ее файлов, например, с помощью Проводника Windows.

При *удалении* таблицы с диска удаляются все ее файлы. В отличие от переименования удаление таблицы можно выполнить посредством любой программы (в том числе и с помощью Проводника Windows).

Ключи и индексы

Как отмечалось, *ключ* представляет собой комбинацию полей, данные в которых однозначно определяют каждую запись в таблице. Простой ключ состоит из одного поля, а составной (сложный) — из нескольких полей. Поля, по которым построен ключ, называют *ключевыми*. В таблице может быть определен только один ключ. Ключ также называют *первичным ключом* или *первичным (главным) индексом*.

Информация о ключе может храниться в отдельном файле или совместно с данными таблицы. Например, в БД Paradox для этой цели используется отдельный файл (ключевой файл или файл главного индекса) с расширением *px*. В БД Access вся информация содержится в одном общем файле с расширением *mdb*. Значения ключа располагаются в определенном порядке. Для каждого значения ключа имеется уникальная ссылка, указывающая на расположение соответствующей записи в таблице (в основном ее файле). Поэтому при поиске записи выполняется не последовательный просмотр всей таблицы, а прямой доступ к записи на основании упорядоченных значений ключа.

Ценой, которую разработчик и пользователь платят за использование такой технологии, является увеличение размера БД вследствие необходимости хранения значений ключа, например, в отдельном файле. Размер этого файла зависит не только от числа записей таблицы (что достаточно очевидно), но и от полей, составляющих ключ. В ключевом файле, кроме ссылок на соответствующие записи таблицы, сохраняются и значения самих ключевых полей. Поэтому при вхождении в состав ключа длинных строковых полей размер ключевого файла может оказаться соизмеримым с размером файла с данными таблицы.

Напомним, что удобным вариантом создания ключа будет использование для него поля соответствующего типа, которое автоматически обеспечивает поддержку уникальности значений. Для таблиц Paradox таким является поле

автоинкрементного типа, еще одним достоинством которого является небольшой размер (4 байта). В то же время в таблицах dBase и InterBase поле подобного типа отсутствует, и программист должен обеспечивать уникальность значений ключа самостоятельно, например, используя специальные генераторы.

Отметим, что при создании и ведении БД правильным подходом считается задание в каждой таблице ключа даже в случае, если на первый взгляд он не нужен.

Индекс, как и ключ, строится по полям таблицы, однако он может допускать повторение значений составляющих его полей — в этом и состоит его основное отличие от ключа. Поля, по которым построен индекс, называют *индексными*. Простой индекс состоит из одного поля, а составной (сложный) — из нескольких полей.

Индексы при их создании именуются. Как и в случае с ключом, в зависимости от СУБД индексы могут храниться в отдельных файлах или совместно с данными. Создание индекса называют *индексированием таблицы*.

Сортировка представляет собой упорядочивание записей по полю или группе полей в порядке возрастания или убывания их значений. Можно сказать, что индекс служит для сортировки таблиц по индексным полям. В частности, в C++ Builder записи набора Table можно сортировать только по индексным полям. Набор данных Query позволяет выполнить средствами SQL сортировку по любым полям, однако и в этом случае для индексированных полей упорядочивание записей выполняется быстрее.

Для одной таблицы можно создать несколько индексов. В каждый момент времени один из них можно сделать текущим, т. е. активным. Даже при существовании нескольких индексов таблица может не иметь текущего индекса (текущий индекс важен, например, при выполнении поиска и сортировки записей набора данных Table).

Ключевые поля обычно автоматически индексируются. В таблицах Paradox ключ также является главным (первичным) индексом, который не именуется. Для таблиц dBase ключ не создается, и его роль выполняет один из индексов. Создание ключа может привести к побочным эффектам. Так, если в таблице Paradox определить ключ, то записи автоматически упорядочиваются по его значениям, что в ряде случаев является нежелательным.

Одной из основных задач БД является обеспечение *быстрого доступа к данным* (поиска данных). Время доступа к данным в значительной степени зависит от используемых для поиска данных методов и способов.

Способы доступа к данным

При выполнении операций с таблицами в C++ Builder используется один из следующих способов доступа к данным:

- навигационный;
- реляционный.

Навигационный способ доступа заключается в обработке каждой отдельной записи таблицы. Этот способ обычно используется в локальных БД или в удаленных БД небольшого размера. Если необходимо обработать несколько записей, то все они обрабатываются поочередно.

Реляционный способ доступа основан на обработке сразу группы записей, при этом если необходимо обработать одну запись, то обрабатывается группа, состоящая из одной записи. Так как реляционный способ доступа основывается на SQL-запросах, его также называют *SQL-ориентированным*. Этот способ доступа ориентирован на выполнение операций с удаленными БД и является предпочтительным при работе с ними, хотя его можно использовать и для локальных БД.

Способ доступа к данным выбирается программистом и зависит от средств доступа к БД, используемых при разработке приложения. Например, в приложениях, создаваемых в C++ Builder, реализацию навигационного способа доступа можно осуществить посредством компонентов Table или Query, а реляционного — с помощью компонента Query.

Таким образом, методы доступа к данным определяются структурой БД, а способы доступа — приложением.

Связь между таблицами

Связи между таблицами можно устанавливать как при создании БД, так и при выполнении приложения, используя средства, предоставляемые СУБД. Связывать можно две или несколько таблиц. В реляционной БД, помимо связанных таблиц, могут быть и отдельные таблицы, не соединенные ни с одной другой таблицей.

Для связывания таблиц используются *поля связи*. Они обязательно должны быть индексированными. В подчиненной таблице для связи с главной таблицей задается индекс, который также называется *внешним ключом*. Состав полей этого индекса должен полностью или частично совпадать с составом полей индекса главной таблицы.

Особенности использования индексов зависят от формата связываемых таблиц. Так, для таблиц dBase индексы строятся по одному полю и нет деления на ключ (главный или первичный индекс) и индексы.

Для организации связи в главной и подчиненной таблицах выбираются индексы, составленные по полям совпадающего типа, например, целочисленного.

Для таблиц Paradox в качестве полей связи главной таблицы должны использоваться поля ключа, а для подчиненной таблицы — поля индекса. Кроме того, в подчиненной таблице обязательно должен быть определен ключ. На рис. 11.13 показана схема связи между таблицами БД Paradox.

Главная таблица			Подчиненная таблица		
Ключ			Индекс		(внешний ключ)
M_Code	• • •		D_Num	D_Code	• • •
Ключевое поле	• • •		Ключевое поле	Индексное поле	• • •

Рис. 11.13. Схема связи между таблицами базы данных Paradox

В главной таблице определен ключ, построенный по полю **M_Code** автоинкрементного типа. В подчиненной таблице определен ключ по полю **D_Num** также автоинкрементного типа и индекс, построенный по полю **D_Code** целочисленного типа. Связь между таблицами устанавливается по полям **D_Code** и **M_Code**. Индекс по полю **D_Code** является внешним ключом. В названия полей включены префиксы, указывающие на принадлежность поля соответствующей таблице. Так, названия полей главной таблицы начинаются с буквы **M** (**Master**), а названия полей подчиненной таблицы начинаются с буквы **D** (**Detail**). Подобное именование полей облегчает ориентацию в их названиях, особенно при большом количестве таблиц.

Как отмечалось, поля связи должны быть индексированными, хотя, строго говоря, это требование не всегда является обязательным. При доступе к данным средствами языка SQL можно связать (соединить) между собой таблицы и по неиндексированным полям. Однако в этом случае скорость выполнения операций будет низкой.

Связь между таблицами определяет отношение подчиненности, при котором одна таблица является главной (родительской, или мастером — **Master**), а вторая — *подчиненной* (дочерней, или детальной — **Detail**). Саму связь (отношение) называют связь «главный-подчиненный», «родительский-дочерний» или «мастер-детальный».

После установления связи между таблицами при перемещении на какую-либо запись в главной таблице в подчиненной таблице автоматически становятся доступными записи, у которых значение поля связи равно значению поля связи текущей записи главной таблицы. Такой отбор записей подчиненной таблицы является своего рода фильтрацией.

Ограничения по установке, изменению полей связи и каскадному удалению записей могут быть наложены на таблицы при их создании. Эти ограничения, наряду с другими элементами, например описаниями полей и индексов, вхо-

дят в структуру таблицы и действуют для всех приложений, которые выполняют операции с БД. Указанные ограничения можно задать при создании или реструктуризации таблицы, например, в среде программы Database Desktop, которая позволяет устанавливать связи между таблицами при их создании.

Ограничения, связанные с установкой, изменением значений полей связи и каскадным удалением записей, могут и не входить в структуру таблицы (таблиц), а реализовываться программным способом. В этом случае программист должен обеспечить:

- организацию связи между таблицами;
- установку значения поля связи подчиненной таблицы (это может также выполняться автоматически);
- контроль (запрет) редактирования полей связи;
- организацию (запрет) каскадного удаления записей.

Например, в случае удаления записи из главной таблицы программист должен проверить наличие соответствующих записей в подчиненной таблице. Если такие записи есть, то необходимо удалить и их или, наоборот, запретить удаление записей из обеих таблиц. И в том, и в другом случае пользователю должно быть выдано предупреждение.

Форматы таблиц

C++ Builder не имеет своего формата таблиц, но поддерживает как собственные два типа локальных таблиц – dBase и Paradox. Каждая из этих таблиц имеет свои особенности.

Таблицы dBase являются одним из первых появившихся форматов таблиц для персональных компьютеров и поддерживаются многими системами, которые связаны с разработкой и обслуживанием приложений, работающих с БД. Основные достоинства таблиц dBase: простота использования и совместимость с большим числом приложений.

Таблицы dBase являются достаточно простыми и используют для своего хранения на дисках относительно мало физических файлов. По расширению файлов можно определить, какие данные они содержат.

- dbf – таблица с данными.
- dbt – данные больших двоичных объектов, или BLOB-данные (Binary Large OBject). К ним относятся двоичные, Memo- и OLE-поля. Memo-поле также называют полем комментариев.
- mdx – поддерживаемые индексы.
- ndx – индексы, непосредственно не поддерживаемые форматом dBase. При использовании таких индексов программист должен обрабатывать их самостоятельно.

Имя поля в таблице dBase должно состоять из букв и цифр и начинаться с буквы. Максимальная длина имени составляет 10 символов. В имена нельзя включать специальные символы и пробел.

К недостаткам таблиц dBase относится то, что они не поддерживают автоматическое использование парольной защиты и контроль целостности связей, поэтому программист должен кодировать эти действия самостоятельно.

Таблицы Paradox являются достаточно развитыми и удобными для создания БД. Можно отметить следующие их достоинства: большое количество типов полей для представления данных различных типов; поддержка целостности данных; организация проверки вводимых данных; поддержка парольной защиты таблиц.

Большой набор типов полей позволяет гибко выбирать тип для точного представления данных, хранимых в базе. Например, для представления числовой информации можно использовать один из пяти числовых типов.

Благодаря своим достоинствам таблицы Paradox используются чаще. В табл. 11.1 содержится список типов полей для таблиц Paradox 7. Для каждого типа приводятся символ, используемый для обозначения этого типа в программе Database Desktop, и описание значений, которые может содержать поле рассматриваемого типа.

Таблица 11.1.

Типы полей таблиц в Paradox 7

Тип	Обозначение	Описание значения
Alpha	A	Строка символов. Длина не более 255 символов
Number	N	Число с плавающей точкой. Диапазон –10307 ... 10308. Точность 15 цифр мантиссы
Money	\$	Денежная сумма. Отличается от типа Number тем, что в значении отображается денежный знак. Обозначение денежного знака зависит от установок Windows
Short	S	Целое число. Диапазон –32 768 ... 32 767
LongInteger	I	Целое число. Диапазон –2 147 483 648 ... 2 147 483 647
BCD	#	Число в двоично-десятичном формате
Date	D	Дата. Диапазон 01.01.1999 до н. э. ... 31.12.9999
Time	T	Время
Timestamp	@	Дата и время
Memo	M	Строка символов. Длина не ограничена. Первые 240 символов хранятся в файле таблицы, остальные в файле с расширением mb
Formatted	F	Строка символов. Отличается от типа Memo тем, что строка может содержать форматированный текст
Graphic	G	Графическое изображение. Форматы BMP, PCX, TIFF, GIF и EPS. При загрузке в поле изображение преобразуется к формату BMP. Для хранения изображения используется файл с расширением mb

Таблица 11.1 (окончание)

Тип	Обозначение	Описание значения
OLE	O	Данные в формате, который поддерживается технологией OLE. Данные хранятся в файле с расширением <i>mb</i>
Logical	L	Логическое значение. Допустимы значения <i>true</i> (истина) и <i>false</i> (ложь). Разрешается использование прописных букв
Autoincrement	+	Автоинкрементное поле. При добавлении к таблице новой записи в поле автоматически заносится значение, на единицу большее, чем в последней добавленной записи. При удалении записи значение ее автоинкрементного поля больше не будет использовано. Значение автоинкрементного поля доступно для чтения и обычно используется в качестве ключевого поля
Binary	B	Последовательность байтов. Длина не ограничена. Байты содержат произвольное двоичное значение. Первые 240 байтов хранятся в файле таблицы, остальные в файле с расширением <i>mb</i>
Bytes	Y	Последовательность байтов. Длина не более 255 байтов

Замечание.

При работе с таблицей в среде программы Database Desktop значения полей типа Graphic, Binary, Memo и OLE не отображаются.

Имя поля в таблице Paradox должно состоять из букв (допускается кириллица) и цифр и начинаться с буквы. Максимальная длина имени составляет 25 символов. В имени можно использовать такие символы, как пробел, #, \$ и некоторые другие. Не рекомендуется использовать символы ., ! и |, т. к. они зарезервированы в C++ Builder для других целей.

При задании ключевых полей они должны быть первыми в структуре таблицы.

Если требуется обеспечить перенос или совместимость данных из таблиц Paradox с таблицами других форматов, желательно выбирать имя поля длиной не более 10 символов и составлять его из латинских букв и цифр.

Определенным недостатком таблиц Paradox является наличие относительно большого количества типов файлов, требуемых для хранения содержащихся в таблице данных. При копировании или перемещении какой-либо таблицы из одного каталога в другой необходимо обеспечить копирование или перемещение всех файлов, относящихся к этой таблице. Файлы таблиц Paradox имеют следующие расширения:

- db — таблица с данными;
- mb — BLOB-данные;
- px — главный индекс (ключ);

- xg^* и yg^* — вторичные индексы;
 - val — параметры для проверки данных и целостности ссылок;
 - tv и fam — форматы вывода таблицы в программе Database Desktop.
- Указанные файлы создаются по мере необходимости; конкретная таблица может не иметь всех приведенных файлов.

Инструменты

Хотя система C++ Builder не имеет своего формата таблиц БД, тем не менее, она обеспечивает развитую поддержку большого количества различных СУБД — как локальных (например, dBase или Paradox), так и промышленных (например, Sybase или InterBase). Средства C++ Builder, предназначенные для работы с БД, можно разделить на два вида:

- инструменты;
- компоненты.

К *инструментам* относятся специальные программы и пакеты, обеспечивающие обслуживание БД вне разрабатываемых приложений.

Компоненты предназначены для создания приложений, осуществляющих операции с БД. Кроме компонентов, C++ Builder предоставляет разработчику специальные объекты, например, объекты типа Field (задают поля таблицы).

Напомним, что в C++ Builder имеется окно Обозревателя дерева объектов (Object TreeView), которое отображает иерархическую структуру объектов текущей формы. При разработке приложений баз данных его удобно использовать для просмотра структуры базы данных и изменения связей между компонентами. Кроме того, в окне Редактора кода имеется вкладка Diagram, служащая для отображения и настройки взаимосвязей между элементами баз данных.

Кроме процессора баз данных BDE, для операций с БД система C++ Builder предлагает следующий набор инструментов.

- BDE Administrator — утилита для настройки различных параметров BDE, настройки драйверов баз данных, создания и удаления драйверов ODBC, создания и обслуживания псевдонимов.
- Database Desktop — программа создания и редактирования таблиц, SQL-запросов и запросов QBE.
- SQL Explorer — Проводник БД, позволяющий просматривать и редактировать БД и словари данных.
- SQL Builder — программа визуального конструирования SQL-запросов.
- SQL Monitor — программа отслеживания порядка выполнения SQL-запросов к удаленным БД.
- Data Pump — программа для переноса данных (схемы базы данных и содержащего) между БД.
- IBConsole — программа для управления удаленными БД.

- InterBase Server Manager — программа для запуска сервера InterBase.
 - SQL Links — драйверы для доступа приложений (с использованием BDE) к удаленным промышленным СУБД, таким как Microsoft SQL Server или Oracle.
 - dbExpress — набор драйверов для доступа к базам данных SQL (InterBase, DB2, Oracle, MSSQL, MySQL).
 - InterBase Server — клиентская и серверная части сервера InterBase.
- Одни инструменты, например, BDE Administrator и SQL Explorer, можно использовать для работы с локальными и удаленными БД, другие, например, IBConsole, — для работы с удаленными БД.

Компоненты приложений для баз данных

Как и другие элементы управления C++ Builder, связанные с БД компоненты делятся на визуальные и невизуальные. *Невизуальные* компоненты предназначены для организации доступа к данным, содержащимся в таблицах. Они представляют собой промежуточное звено между данными таблиц БД и визуальными компонентами.

Визуальные компоненты используются для создания интерфейской части приложения. С их помощью пользователь может выполнять такие операции с таблицами БД, как просмотр или редактирование данных. Визуальные компоненты также называют *элементами, чувствительными к данным*.

Компоненты, используемые для работы с БД, находятся на страницах Data Access, Data Controls, dbExpress, DataSnap, BDE, ADO, InterBase, Decision Cube, QReport и InterBase Admin Палитры компонентов. Некоторые компоненты предназначены специально для работы с удаленными БД в архитектуре «клиент-сервер». Рассмотрим важнейшие компоненты, расположенные на наиболее часто используемых страницах при работе с базами данных.

На странице Data Access (рис. 11.14) находятся невизуальные компоненты, предназначенные для организации доступа к данным, например:

- DataSource (источник данных);
- ClientDataSet (клиентский набор данных);
- DataSetProvider (провайдер набора данных);
- XML Transform (преобразователь документа XML в пакет данных и обратно).



Рис. 11.14. Страница Data Access

На странице **Data Controls** (рис. 11.15) расположены визуальные компоненты, предназначенные для управления данными:

- **DBGrid** (сетка, или таблица);
- **DBNavigator** (навигационный интерфейс);
- **DBText** (надпись);
- **DBEdit** (однострочный редактор, или поле редактирования);
- **DBMemo** (многострочный редактор, или панель редактирования);
- **DBImage** (графическое изображение);
- **DBListBox** (список);
- **DBComboBox** (комбинированный список);
- **DBCheckBox** (флажок);
- **DRadioButton** (группа переключателей);
- **DBLookupListBox** (список, формируемый по полю другого набора данных);
- **DBLookupComboBox** (комбинированный список, формируемый по полю другого набора данных);
- **DBRichEdit** (полнофункциональный тестовый редактор, или поле редактирования);
- **DBCtrlGrid** (модифицированная сетка);
- **DBChart** (диаграмма).



Рис. 11.15. Страница Data Controls

Страница **BDE** (рис. 11.16) содержит компоненты, предназначенные для управления данными с использованием BDE:

- **Table** (набор данных, основанный на таблице БД);
- **Query** (набор данных, основанный на SQL-запросе);
- **StoredProc** (вызов хранимой процедуры сервера);
- **DataBase** (соединение с БД);
- **Session** (текущий сеанс работы с БД);
- **BatchMove** (выполнение операций над группой записей);
- **UpdateSQL** (изменение набора данных, основанного на SQL-запросе или хранимой процедуре);
- **NestedTable** (вложенная таблица)
- **BDEClientDataset** (клиентский набор данных)



Рис. 11.16. Страница BDE

На странице **ADO** (рис. 11.17) расположены компоненты, предназначенные для управления данными с использованием технологии ADO (Active Data Objects):

- **ADOCConnection** (соединение);
- **ADOCCommand** (команда);
- **ADODaDataSet** (набор данных);
- **ADOTable** (набор данных Table);
- **ADOQuery** (набор данных Query);
- **ADOSStoredProc** (вызов хранимой процедуры сервера);
- **RDSConnection** (соединение RDS).

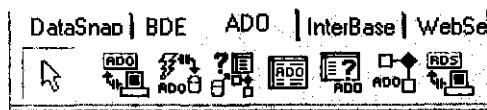


Рис. 11.17. Страница ADO

Замечание

Соединение RDS служит для управления передачей объекта Recordset от одного процесса (компьютера) к другому при создании серверных приложений.

На странице **QReport** (рис. 11.18) находятся компоненты, предназначенные для построения отчетов:

- **QuickRep** (отчет);
- **QRSubDetail** (полоса отчета для таблиц, связанных отношением «главный-подчиненный»);
- **QRStringsBand** (строковая полоса отчета);
- **QRBand** (полоса отчета);
- **QRChildBand** (дочерняя полоса отчета);
- **QRGroup** (группа);
- **QRLabel** (надпись);
- **QRDBText** (текстовое поле набора данных);
- **QRExpr** (выражение);
- **QRSysData** (системная информация);
- **QRMemo** (многострочный текст);
- **QREExprMemo** (многострочное выражение);
- **QRRichText** (форматированный текст);
- **QRDBRichText** (форматированный текст поля набора данных);
- **QRShape** (геометрическая фигура);
- **QRIImage** (графический образ);

- QRDBImage (графический образ поля набора данных);
- QRCompositeReport (составной отчет);
- QRPreview (окно просмотра отчета);
- QRTextFilter (текстовый фильтр);
- QRCSVFilter (CSV-фильтр);
- QRHTMLFilter (HTML-фильтр);
- QRChart (диаграмма).

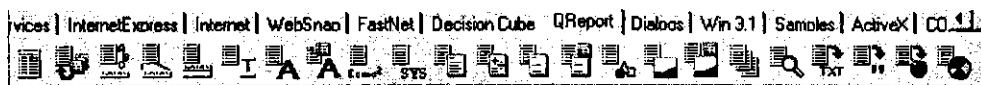


Рис. 11.18. Страница QReport

Имена многих компонентов, предназначенных для работы с данными, содержат префиксы, например, DB, IB или QR. Префикс DB означает, что визуальный компонент связан с данными и используется для построения интерфейской части приложения. Такие компоненты размещаются в форме и предназначены для управления данными со стороны пользователя. Префикс QR означает, что компонент используется для построения отчетов QReport.

Исключения при работе с базами данных

В дополнение к рассмотренным ранее исключениям, специально для операций, связанных с работой приложений для БД, система C++ Builder предоставляет следующие дополнительные классы исключений:

- **EDatabaseError** — ошибка БД; потомки этого класса:
 - **EDBEngineError** — ошибка BDE (для локальных БД и сетевых БД архитектуры «файл-сервер»);
 - **EDBCClient** — ошибка в приложении клиента (для сетевых БД архитектуры «клиент-сервер»); код ошибки возвращается BDE, ADO, dbExpress или другим механизмом доступа к данным;
 - **EUpdateError** — ошибка, возникающая при обновлении записей;
- **EDBEEditError** — введенное в поле значение не соответствует типу поля.

Класс **EDatabaseError** предназначен для обработки ошибок, возникающих при выполнении операций с набором данных (класс **TDataSet** и его потомки, в первую очередь, **TTable** и **TQuery**). Этот класс производится непосредственно от класса **Exception**. Исключение класса **EDatabaseError** генерируется, например, при попытке открыть набор данных, связанный с отсутствующей таблицей, или изменить запись набора данных, который находится в режиме просмотра.

Класс **EDBEngineError** предназначен для обработки ошибок, возникающих при работе с процессором баз данных BDE на локальном компьютере. Класс **TDBError** содержит информацию об исключении **EDBEngineError**, определяемую следующими свойствами:

- **Message** типа **String** (текст сообщения, характеризующего возникшую ошибку);
- **ErrorCode** типа **DBIResult** (код ошибки), тип **DBIResult** соответствует типу **Word**;
- **Category** типа **Byte** (категория исключения);
- **SubCode** типа **Byte** (группа, или подкод, исключения);
- **NativeError** типа **Longint** (код ошибки, возвращаемой сервером) – если значение этого свойства равно нулю, то исключение произошло не на сервере.

Класс **EDBClient** отличается от класса **EDBEngineError** в основном тем, что предназначен для обработки ошибок, возникающих при работе с различными механизмами доступа к данным (не только с процессором баз данных BDE) в операциях с сетевыми базами архитектуры «клиент-сервер».

Генерируемые при работе с БД исключения обрабатывают глобальные и локальные обработчики. Кроме того, используемые для доступа к данным компоненты имеют специальные события для обработки исключений.

Например, для набора данных **Table** такими событиями являются:

- **OnEditError** (ошибка редактирования или вставки записи);
- **OnPostError** и **OnUpdateError** (ошибка закрепления изменений в записи);
- **OnDeleteError** (ошибка удаления записи).

Класс **EDBEeditError** используется в случаях, когда вводимые в поле данные несовместимы с маской ввода, заданной с помощью свойства **EditMask** этого поля. Отметим, что для проверки вводимых в поле значений можно также использовать события **OnSetText**, **OnValidate** и **OnChange**.

11.7. Создание таблиц базы данных

Работа по созданию информационной системы включает два основных этапа: создание БД и создание приложения. Продемонстрируем возможности C++ Builder по работе с БД на примере создания простой информационной системы. Эту информационную систему можно разработать даже без написания кода: все необходимые операции выполняются с помощью программы Database Desktop, Конструктора формы и Инспектора объектов.

В простейшем случае БД состоит из одной таблицы. Если таблицы уже имеются, то первый этап не выполняется. Отметим, что совместно с C++ Builder поставляется большое количество примеров приложений, в том числе и приложений БД. Файлы таблиц для этих приложений находятся в каталоге c:\Program Files\Common Files\Borland Shared\Data. Готовые таблицы также можно использовать для своих приложений.

Для работы с таблицами БД при проектировании приложения удобно использовать программу Database Desktop, которая позволяет:

- создавать таблицы;
- изменять структуры;
- редактировать записи.

Кроме того, с помощью Database Desktop можно выполнять и другие действия над БД (создание, редактирование и выполнение визуальных и SQL-запросов, операции с псевдонимами). Отметим, что большинство рассматриваемых действий по управлению структурой таблицы можно выполнить также программно.

Процесс создания новой таблицы начинается с вызова команды **File\New\Table** (Файл\Новая\Таблица) и происходит в интерактивном режиме. При этом разработчик должен:

- выбрать формат (тип) таблицы;
- задать структуру таблицы.

В начале создания новой таблицы в окне **Create Table** (Создание таблицы) выбирается ее формат. По умолчанию предлагается формат таблицы Paradox версии 7, который мы и будем использовать. Для таблиц других форматов, например dBase IV, действия по созданию таблицы практически не отличаются.

После выбора формата таблицы появляется окно определения структуры таблицы (рис. 11.19), в котором выполняются следующие действия:

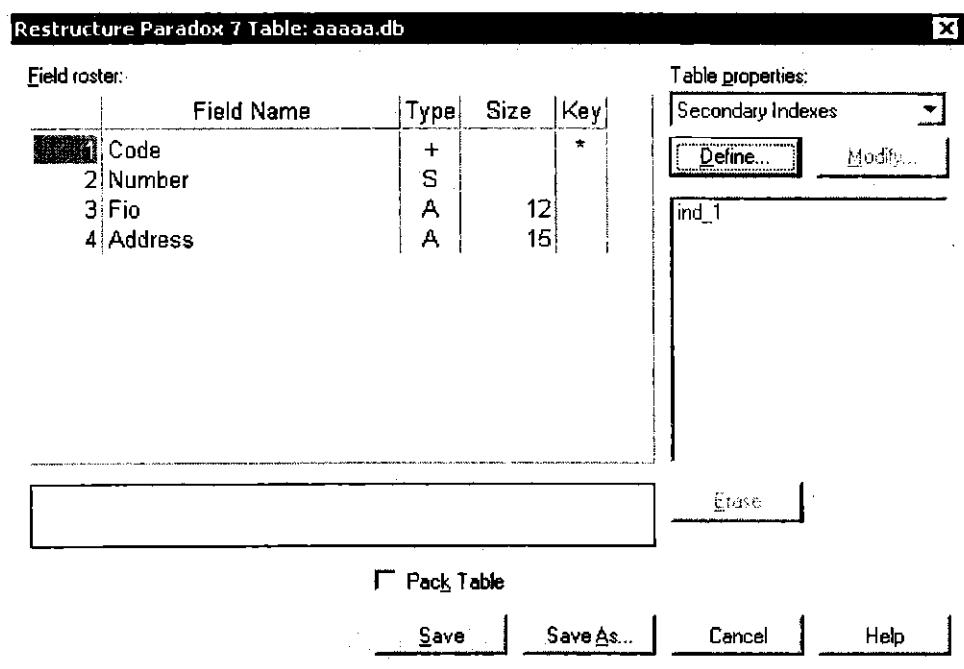


Рис. 11.19. Определение структуры таблицы

- описание полей;
- задание ключа;
- задание индексов;
- определение ограничений на значения полей;
- определение условий (ограничений) ссылочной целостности;
- задание паролей;
- задание языкового драйвера;
- задание таблицы для выбора значений.

В этом списке обязательным является только первое действие, т. е. каждая таблица должна иметь хотя бы одно поле. Остальные действия выполняются при необходимости. Часть действий, такие как задание ключа и паролей, производится только для таблиц определенных форматов, например, для таблиц Paradox.

После определения структуры таблицы ее необходимо сохранить, нажав кнопку **Save As** и указав расположение таблицы на диске и ее имя. В результате на диск записывается новая таблица, первоначально пустая, при этом все необходимые файлы создаются автоматически.

Описание полей

Центральной частью окна определения структуры таблицы является список **Field roster** (Список полей), в котором указываются поля таблицы. Для каждого поля задаются:

- имя поля — в столбце **Field Name**;
- тип поля — в столбце **Type**;
- размер поля — в столбце **Size**.

Имя поля вводится по правилам, установленным для выбранного формата таблиц.

Тип поля можно задать, непосредственно указав соответствующий символ, например, **A** для символьного или **I** для целочисленного поля, или выбрать его в списке, раскрываемом нажатием клавиши <пробел> или щелчком правой кнопки мыши в столбце **Type**. Список содержит все типы полей, допустимые для заданного формата таблицы. В списке подчеркнуты символы, используемые для обозначения соответствующего типа, при выборе типа эти символы автоматически заносятся в столбец **Type**.

Размер поля задается не всегда, необходимость его указания зависит от типа поля. Для полей определенного типа, например, автоинкрементного (+) или целочисленного (!), размер поля не задается. Для поля строкового типа размер определяет максимальное число символов, которые могут храниться в поле.

Добавление к списку полей новой строки выполняется переводом курсора вниз на несуществующую строку, в результате чего эта строка появляется в конце списка. Вставка новой строки между существующими строками с уже

описанными полями выполняется нажатием клавиши <Insert>. Новая строка вставляется перед строкой, в которой расположен курсор. Для удаления строки необходимо установить курсор на эту строку и нажать комбинацию клавиш <Ctrl>+<Delete>.

Ключ создается указанием его полей. Для указания ключевых полей в столбце ключа (**Key**) нужно установить символ *, переведя в эту позицию курсор и нажав любую алфавитно-цифровую клавишу. При повторном нажатии клавиши отметка принадлежности поля ключу снимается. В структуре таблицы ключевые поля должны быть первыми, т. е. верхними в списке полей. Часто для ключа используют автоинкрементное поле (см. рис. 11.19).

Напомним, что для таблиц Paradox ключ также называют первичным индексом (Primary Index), а для таблиц dBase ключ не создается, и его роль выполняет один из индексов.

Для выполнения остальных действий по определению структуры таблицы используется комбинированный список **Table properties** (Свойства таблицы), содержащий следующие пункты:

- **Validity Checks** (проверка правильности ввода значений полей) — выбирается по умолчанию;
- **Table Lookup** (таблица выбора);
- **Secondary Indexes** (вторичные индексы);
- **Referential Integrity** (ссылочная целостность);
- **Password Security** (пароли);
- **Table Language** (язык таблицы, языковой драйвер);
- **Dependent Tables** (подчиненные таблицы).

После выбора какого-либо пункта этого списка в правой части окна определения структуры таблицы появляются соответствующие элементы, с помощью которых выполняются дальнейшие действия.

Состав данного списка зависит от формата таблицы. Так, для таблицы dBase он содержит только пункты **Indexes** и **Table Language**.

Задание индексов

Задание индекса сводится к определению:

- состава полей;
- параметров;
- имени.

Эти элементы устанавливаются или изменяются при выполнении операций создания, изменения и удаления индекса. Напомним, что для таблиц Paradox индекс называют также вторичным индексом.

Для выполнения операций, связанных с заданием индексов, необходимо выбрать пункт **Secondary Indexes** (Вторичные индексы) списка **Table properties** (Свойства таблицы), при этом под списком появляются кнопки **Define** (Определить) и **Modify** (Изменить), список индексов и кнопка **Erase**

(Удалить). В списке индексов выводятся имена созданных индексов, на рис. 11.19 это индекс ind_1.

Создание нового индекса начинается с нажатия кнопки Define, которая всегда доступна. Она открывает окно Define Secondary Index (Задание вторичного индекса), в котором задаются состав полей и параметры индекса (рис. 11.20).

В списке Fields окна выводятся имена всех полей таблицы, включая и те, которые недопустимы в составе индекса, например, графическое поле или поле комментария. В списке Indexed fields (Индексные поля) содержатся поля, которые включаются в состав создаваемого индекса. Перемещение полей между списками выполняется выделением нужного поля (полей) и нажатием расположенных между этими списками кнопок с изображением горизонтальных стрелок. Имена полей, которые нельзя включать в состав индекса, выделяются в левом списке серым цветом. Поле не может быть повторно включено в состав индекса, если оно уже выбрано и находится в правом списке.

Замечание.

При работе с записями индексные поля обрабатываются в порядке следования этих полей в составе индекса. Это нужно учитывать при указании порядка полей в индексе.

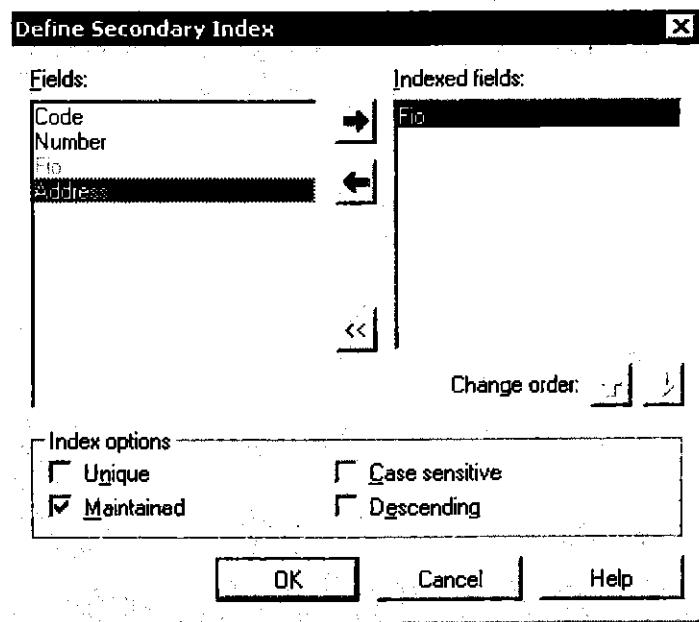


Рис. 11.20. Окно задания индекса

Изменить порядок следования полей в индексе можно с помощью кнопок с изображением вертикальных стрелок, имеющих общее название **Change order** (Изменить порядок). Для перемещения поля (полей) необходимо его (их) выделить и нажать нужную кнопку.

Флажки, расположенные в нижней части окна задания индекса, позволяют указать следующие параметры индекса:

- **Unique** — индекс требует для составляющих его полей уникальных значений;
- **Maintained** — задается автоматическое обслуживание индекса;
- **Case sensitive** — для полей строкового типа учитывается регистр символов;
- **Descending** — сортировка выполняется в порядке убывания значений.

Так как у таблиц dBase нет ключей, для них использование параметра **Unique** является единственной возможностью обеспечить уникальность записей на физическом уровне (уровне организации таблицы), не прибегая к программированию.

После задания состава индексных полей и нажатия кнопки **OK** появляется окно **Save Index As**, в котором нужно указать имя индекса. Для удобства обращения к индексу в его имя можно включить имена полей, указав какой-нибудь префикс, например *ind*. Нежелательно образовывать имя индекса только из имен полей, т. к. для таблиц Paradox подобная система именования используется при автоматическом образовании имен для обозначения ссылочной целостности между таблицами. После повторного нажатия кнопки **OK** сформированный индекс добавляется к таблице, и его имя появляется в списке индексов.

Созданный индекс можно изменить, определив новый состав полей, параметров и имени индекса. Изменение индекса практически не отличается от его создания. После выделения индекса в списке и нажатия кнопки **Modify** снова открывается окно задания индекса (см. рис. 11.20). При нажатии кнопки **OK** появляется окно сохранения индекса, содержащее имя изменяемого индекса, которое можно исправить или оставить прежним.

Для удаления индекса его нужно выделить в списке индексов и нажать кнопку **Erase**. В результате индекс удаляется без предупреждающих сообщений.

Кнопки **Modify** и **Erase** доступны, только если индекс выбран в списке.

Задание ограничений на значения полей

Задание ограничений на значения полей заключается в указании для полей:

- требования обязательного ввода значения;
- минимального значения;
- максимального значения;
- значения по умолчанию;
- маски ввода.

Установленные ограничения задаются на физическом уровне (уровне таблицы) и действуют для любых программ, выполняющих операции с таблицей: как для программ типа Database Desktop, так и для приложений, создаваемых в C++ Builder. Дополнительно к этим ограничениям или вместо них в приложении можно задать программные ограничения.

Для выполнения операций, связанных с заданием ограничений на значения полей, нужно выбрать пункт **Validity Checks** (Проверка значений) комбинированного списка **Table properties** (см. рис. 11.19).

Задание ссылочной целостности

Понятие ссылочной целостности относится к связанным таблицам и проявляется в следующих вариантах взаимодействия таблиц:

- запрещается изменять поле связи или удалять запись главной таблицы, если для нее имеются записи в подчиненной таблице;
- при удалении записи в главной таблице автоматически удаляются соответствующие ей записи в подчиненной таблице (каскадное удаление).

Для выполнения операций, связанных с заданием ссылочной целостности, необходимо выбрать пункт **Referential Integrity** комбинированного списка **Table properties** (см. рис. 11.19).

Задание паролей

Пароль позволяет задать права доступа пользователей (приложений) к таблице. Если для таблицы установлен пароль, то он будет автоматически запрашиваться при каждой попытке открытия таблицы.

Пароль действует на физическом уровне и его действие распространяется на все программы, выполняющие доступ к таблице: как на программы типа Database Desktop, так и на создаваемые приложения C++ Builder.

Для выполнения операций, связанных с заданием пароля, нужно выбрать строку **Password Security** в комбинированном списке **Table properties** окна определения структуры таблицы (см. рис. 11.19).

Задание языкового драйвера

Для задания языкового драйвера нужно выбрать пункт **Table Language** (Язык таблицы) комбинированного списка **Table properties** окна определения структуры таблицы (см. рис. 11.19).

Изменение структуры таблицы

Структуру существующей таблицы можно изменить, выполнив команду **Table\Restructure** после предварительного выбора таблицы в окне программы Database Desktop. В результате открывается окно определения структуры таблицы, и дальнейшие действия не отличаются от действий, выполняемых при создании таблицы.

При изменении структуры таблицы с ней не должны работать другие приложения, в том числе C++ Builder. Поэтому предварительно необходимо закрыть C++ Builder или приложение, в котором компоненты **Table** связаны с перестраиваемой таблицей. Другим вариантом является отключение активности компонентов **Table**, связанных с перестраиваемой таблицей, для чего свойству **Active** этих компонентов через Инспектор объектов устанавливается значение **false**.

Переименование таблицы следует выполнять из среды программы Database Desktop, а не из среды Windows, например, с помощью Проводника. Для этого при работе со структурой таблицы можно нажать кнопку **Save as** и задать новое имя таблицы. В результате в указанном каталоге диска появятся все необходимые файлы таблицы. При этом старая таблица также сохраняется. Информация о назывании таблицы используется внутри ее файлов, поэтому простое переименование всех файлов таблицы приведет к ошибке при попытке доступа к ней.

Если необходимо просто ознакомиться со структурой таблицы, то выполняется команда **Table\Info Structure**. В результате появляется окно определения структуры таблицы, но элементы, с помощью которых в структуру таблицы могут быть внесены изменения, заблокированы. Просмотр структуры возможен также для таблицы, с которой связаны другие приложения.

11.8. Создание приложения BDE

Для примера рассмотрим создание приложения, использующего механизм доступа BDE и позволяющего перемещаться по записям таблицы БД, просматривать и редактировать поля, удалять записи из таблицы, а также вставлять новые. Файл проекта приложения обычно не требует от разработчика выполнения каких-либо действий. Поэтому при создании приложения главной задачей является конструирование форм, в простейшем случае — одной формы.

Вид формы приложения на этапе проектирования показан на рис. 11.21, где в форме размещены компоненты **Table1**, **DataSource1**, **DBGrid1** и **DBNavigator1**.

Компонент **Table1** обеспечивает взаимодействие с таблицей БД. Для связи с требуемой таблицей нужно установить в соответствующие значения свойство **DataBaseName**, указывающее путь к БД, и свойство **TableName**, указывающее имя таблицы. После задания таблицы для открытия набора данных свойству **Active** должно быть установлено значение **true**.

Свойству **Active** нужно устанавливать значение **true** после задания таблицы БД, т. е. после установки нужных значений свойств **DataBaseName** и **TableName**.

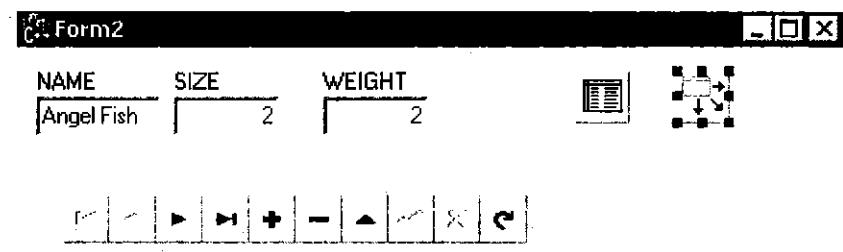


Рис. 11.21. Форма приложения для работы с БД

Имя таблицы лучше выбирать в раскрывающемся списке в поле значения свойства **TableName**. Если путь к БД (свойство **DataBaseName**) задан правильно, то в этом списке отображаются главные файлы всех доступных таблиц.

В рассматриваемом приложении использована таблица животных, входящая в состав поставляемых с C++ Builder примеров, ее главный файл — **Animals.dbf**. Файлы этой и других таблиц примеров находятся в каталоге, путь к которому указывает псевдоним **dbdemos**. Настройка псевдонима может быть выполнена с помощью программы **BDE Administrator**.

Компонент **DataSource1** является промежуточным звеном между компонентом **Table1**, соединенным с реальной таблицей БД, и визуальными компонентами **DBGrid1** и **DBNavigator1**, с помощью которых пользователь взаимодействует с этой таблицей. На компонент **Table1**, с которым связан компонент **DataSource1**, указывает свойство **DataSet** последнего.

Компонент **DBGrid1** отображает содержимое таблицы БД в виде сетки, в которой столбцы соответствуют полям, а строки — записям таблицы. По умолчанию пользователь может просматривать и редактировать данные. Компонент **DBNavigator1** позволяет пользователю перемещаться по таблице, редактировать, вставлять и удалять записи. Компоненты **DBGrid1** и **DBNavigator1** связываются со своим источником данных — компонентом **DataSource1** — через свойства **DataSource**.

Взаимосвязь компонентов приложения и таблицы БД и используемые при этом свойства компонентов показаны на рис. 11.22.

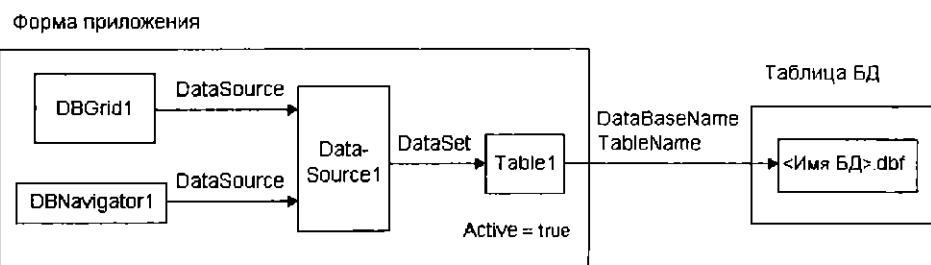


Рис. 11.22. Взаимосвязь компонентов приложения и таблицы БД

Разрабатывая приложение, можно задавать значения всех свойств компонентов с помощью Инспектора объектов. При этом требуемые значения либо непосредственно вводятся в поле, либо выбираются в раскрывающихся списках. В последнем случае приложение создается с помощью мыши и не требует набора каких-либо символов на клавиатуре. В табл. 11.2 приведены компоненты, используемые для работы с таблицей БД, их основные свойства и значения этих свойств.

В дальнейшем при организации приложений, использующих механизм доступа BDE, предполагается, что названные компоненты связаны между собой именно таким образом, и свойства, с помощью которых эта связь осуществляется, не рассматриваются.

Таблица 11.2.

Значения свойств компонентов

Компонент	Свойства	Значения
Table1	DataBaseName	dbdemos
	TableName	Animals.dbf
	Active	true
DataSource1	DataSet	Table1
DBGrid1	DataSource	DataSource1
DBNavigator1	DataSource	DataSource1

Для автоматизации процесса создания формы, использующей компоненты для операций с БД, можно вызвать Database Form Wizard (Мастер форм баз данных). Этот Мастер расположен на странице Business Хранилища объектов.

Мастер позволяет создавать формы для работы с отдельной таблицей и со связанными таблицами, при этом можно использовать наборы данных Table или Query.

11.9. Работа с отчетами

Отчет представляет собой печатный документ, в котором содержит такие же данные, как получаемые в результате выполнения запроса к БД. Можно выделить следующие виды отчетов: простой; с группированием данных; для таблиц, связанных отношением «главный-подчиненный»; составной, объединяющий несколько разных отчетов. В C++ Builder для создания отчетов служит генератор отчетов QuickReport, содержащий большой набор компонентов. Мы рассмотрим средства создания отчета и приведем пример создания простого отчета.

Компоненты отчета

Компоненты, предназначенные для создания отчетов, находятся на странице **QReport** Палитры компонентов. Большинство компонентов отчета являются визуальными. Многие из них мало отличаются от аналогичных компонентов страниц **Standard**, **Additional** и **Data Controls**. Например, компоненту **QRImage** соответствуют компоненты **Image** и **DBImage**.

Главным элементом отчета является **компонент-отчет** **QuickRep**, представляющий собой основу, на которой размещаются другие компоненты. Компонент **QuickRep** обычно размещается на отдельной форме, предназначенной для создания отчета. По умолчанию он имеет имя **QuickRep1**. Если на форме размещается другой отчет (обычно так не делается), он получает имя **QuickRep2** и т. д.

Компонент **QuickRep** при помещении его на форму имеет вид страницы формата А4, первоначально отображаемой в натуральную величину. При разработке приложения можно изменить масштаб страницы и размещенных на ней компонентов с помощью свойства **Zoom** типа **int**. Значение этого свойства устанавливается в процентах, по умолчанию 100%.

Отчет можно поместить на любую форму приложения, например, на главную. В этом случае отчет (страница) создает своего рода фон, на котором расположены управляющие элементы формы.

Компонент **QuickRep** связывается с набором данных **Table** или **Query**, для которого создается отчет, с помощью свойства **DataSet**. При этом набор данных **Query** может содержать записи, выбранные из разных таблиц. При печати отчета в процессе выполнения приложения набор данных должен быть открыт. Во время построения отчета можно использовать специально создаваемый набор данных и размещать его на форме, при этом источник данных **DataSource** не требуется. На практике компонент **QuickRep** обычно связывается с набором данных, записи которого отображаются на форме в визуальных компонентах. В этом случае в отчет попадают записи, удовлетворяющие, например, критерию фильтрации и/или сортировки, задаваемому пользователем.

Пример. Связывание отчета с набором данных.

```
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    QuickRep1->DataSet=Form1->Table1;
}
```

Отчет **QuickRep1**, находящийся на форме **Form2**, при обработке события **OnCreate** этой формы связывается с набором данных **Table1**, расположенным на форме **Form1**. В файле реализации (исходного кода) модуля формы **Form2** следует поместить директиву препроцессора **#include «Unit1.h»** для подключения заголовочного файла модуля формы **Form1**.

Отчет состоит из отдельных полос — составных частей отчета, которые определяют содержание и вид созданного документа. Полоса представляет собой элемент отчета. Каждый элемент размещается на своем месте и предназначен для отображения соответствующих компонентов отчета и вывода данных. Возможный вид отчета при разработке показан на рис. 11.23.

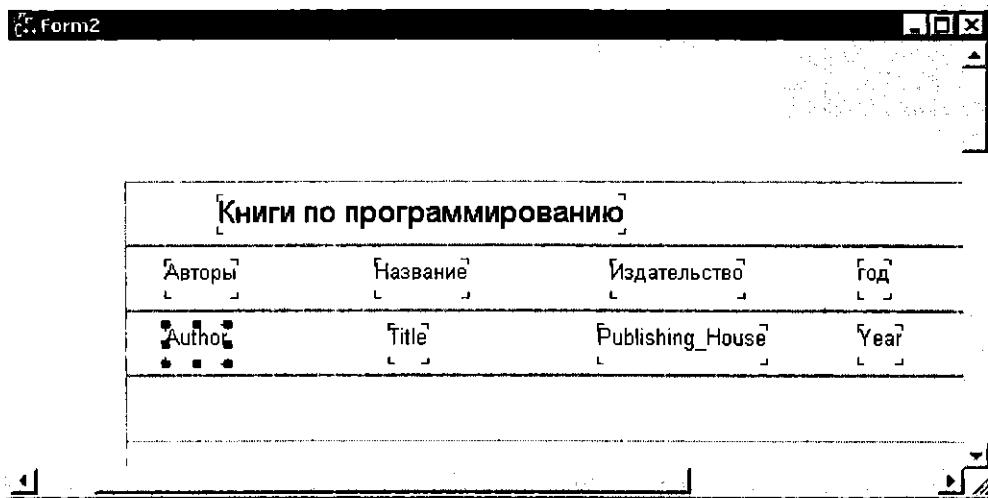


Рис. 11.23. Вид отчета при разработке

Конструирование отчета в основном аналогично конструированию формы. Управлять наличием полос в отчете можно с помощью свойства **Bands** множественного типа **TQuickRepBands**. При разработке приложения включение/отключение полосы выполняется установкой логического значения соответствующего подсвойства свойства **Bands**, например, для полосы области данных отчета этим подсвойством является **HasDetail**. С помощью этого свойства в простой отчет можно включать следующие полосы:

- **HasPageHeader** — верхний колонтитул;
- **HasTitle** — заголовок отчета;
- **HasColumnHeader** — заголовки столбцов;
- **HasDetail** — область данных;
- **HasSummary** — итог отчета;
- **HasPageFooter** — нижний колонтитул.

Перечисленные полосы можно также вставлять в отчет с помощью компонента полосы **QRBand**, при этом тип вставляемой полосы устанавливается через свойство **BandType** этого компонента.

Параметры страницы отчета определяет свойство **Page** типа **TQRPage**, через подсвойства которого можно настраивать, например, следующие характеристики:

- **PaperSize** — формат страницы (по умолчанию A4);
- **TopMargin**, **BottomMargin**, **LeftMargin** и **RightMargin** — размер верхнего, нижнего, левого и правого полей, соответственно.

Единицы измерения страниц, полос, полей, а также других элементов отчета определяет свойство **Unit** типа **TQRUnits**, принимающее, к примеру, следующие значения:

- **Inches** — дюймы;
- **MM** — миллиметры (по умолчанию);
- **Characters** — автоматическая настройка в соответствии с размером символов, установленным свойством **QuickRep->Font->Size**.

При необходимости разработчик может изменить параметры страницы, а также многие другие параметры отчета (например, шрифт по умолчанию) с помощью Инспектора объектов или в диалоговом окне **Report Setting** (Установки отчета) установки параметров отчета. Оно вызывается командой **Report Setting** контекстного меню страницы отчета или двойным щелчком на странице отчета.

Страница отчета может иметь рамку, параметры которой задает свойство **Frame** типа **TQRFrame**.

С помощью свойства **PrinterSetting** типа **TQuickRepPrinterSettings** устанавливаются параметры принтера, например: **FirstPage** и **LastPage** — номер первой и последней печатаемой страницы. Параметры принтера можно устанавливать также с помощью стандартных диалогов **PrintDialog** и **PrinterSetupDialog** или метода **PrinterSetup**.

Отчет характеризуется тремя параметрами, которые задаются в свойстве **Options** множественного типа **TQuickReportOptions**:

- **FirstPageHeader** — печать верхнего колонтитула на первой странице отчета;
- **LastPageFooter** — печать нижнего колонтитула на последней странице отчета;
- **Compression** — отчет сохраняется в сжатом формате, при этом уменьшается объем занимаемой памяти, но снижается быстродействие.

Свойство **PrintIfEmpty** типа **bool** определяет, выводить ли данные отчета для пустого набора данных. По умолчанию свойство имеет значение **true**, и отчет печатается, даже если в наборе данных нет ни одной записи. Это удобно, например, при печати бланков. Если свойству **PrintIfEmpty** установить значение **false**, то для пустого набора данных отчет не выводится, точнее, выводится пустая страница. То есть при отсутствии записей отсутствует не только область данных, но и ряд других полос, например, заголовок отчета.

Как правило, свойства отчета и его отдельных элементов устанавливаются при создании отчета на этапе разработки приложения.

Для печати отчета предназначен метод **Print**, сразу после вызова которого отчет подготавливается к печати и направляется на установленный в системе

принтер. Дополнительных подтверждений от пользователя не требуется. При необходимости текущий принтер и его параметры устанавливаются до начала печати, например, с помощью вызова метода `PrinterSetup`. Метод `Print` может вызываться, например, при нажатии кнопки **Печать**, расположенной на форме, с которой пользователь работает.

Если компонент-отчет `QuickRep` связан с набором данных, записи которого выводятся в сетке `DBGrid` формы, то порядок записей отчета соответствует порядку записей, видимых пользователем на форме. После отбора (фильтрации) записей и/или сортировки при нажатии кнопки **Печать** происходит вывод отчета, причем учитывается новый состав и порядок следования записей.

При формировании отчета изменяется положение указателя текущей записи, поэтому при необходимости разработчик должен предусмотреть запоминание и восстановление положения указателя, например, с помощью закладки.

Пример. Процедура предварительного просмотра и печати отчета.

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
TBookmark bm;
bm=Table1->GetBookmark();
// Просмотр отчета
Form2->QuickRep1->Preview();
// Печать отчета
Form2->QuickRep1->Print();
Table1->GotoBookmark(bm);
Table1->FreeBookmark(bm);
}
```

Закладка `bm` используется для запоминания и восстановления положения указателя текущей записи. В файле исходного кода модуля формы `Form1`, из которой выполняется печать отчета, следует указать директиву препроцессора `#include "Unit2.h"` для подключения заголовочного файла модуля формы `Form2`, в которой размещен компонент отчета.

При печати отчета генерируются события `BeforePrint` и `AfterPrint` типа `TQRBeforePrintEvent`. С помощью обработчика первого события можно задать действия, выполняемые непосредственно перед печатью отчета, а с помощью обработчика второго события — действия, выполняемые сразу после окончания печати.

В процессе выполнения приложения для предварительного просмотра отчета перед печатью служит метод `Preview`, вызывающий окно просмотра. В этом окне можно выполнить следующее: просмотреть отчет в различных

масштабах; сохранить отчет в файле; загрузить предварительно сохраненный отчет; направить отчет в печать.

Возможности метода **Preview** превосходят возможности метода **Print**, поэтому чаще выполняют именно предварительный просмотр документа, а не печать, что удобно и при отладке приложения. Печатать отчет можно непосредственно из окна предварительного просмотра.

На этапе разработки приложения также можно просмотреть отчет, выполнив команду **Preview** (Просмотр) контекстного меню отчета. Внешний вид отчета будет таким же, как при печати или в окне просмотра при выполнении приложения, за исключением отсутствия значений вычисляемых полей.

Разработчик имеет возможность создать свое окно предварительного просмотра отчета, используя компонент **QRPreview**.

Полоса отчета (компонент **QRBand**) является основной составной частью (элементом) отчета, на которой размещаются другие его компоненты.

Тип полосы определяется свойством **BandType** типа **TQRBandType**, которое может принимать, к примеру, следующие значения:

- **rbTitle** — заголовок отчета (печатается в начале отчета под верхним колонтитулом);
- **rbPageHeader** — верхний колонтитул, который печатается сверху на каждой странице, в том числе на первой, если включен (по умолчанию) параметр **FirstPageFooter** свойства **Options** компонента отчета; если этот параметр выключен, то верхний колонтитул на первом листе не печатается;
- **rbDetail** — данные записей набора данных; выводятся для каждой записи набора данных;
- **rbChild** — дочерняя полоса, печатаемая после полосы, с которой она связана.

При установке типа полосы она автоматически размещается на своем месте в отчете и выравнивается по ширине страницы отчета с учетом левого и правого полей. Разработчик не имеет возможности переместить полосу на другое место с помощью мыши. Изменить ширину полосы можно косвенно, изменяя размеры страницы и полей. Высота полосы меняется путем передвижения мышью верхней или нижней рамки полосы или через установку значения свойства **Height** в Инспекторе объектов.

Добавить новую полосу к отчету можно следующими двумя способами:

- поместить компонент **QRBand** в отчет и присвоить требуемое значение свойству **BandType**;
- установить значение **true** соответствующему подсвойству свойства **Bands** компонента **QuickRep**, при этом к отчету добавляется полоса, ее свойству **BandType** автоматически устанавливается нужное значение.

При создании в отчет нужно включать не более одной полосы каждого вида, так как при печати отчета «лишние» полосы одного и того же вида учитываться не будут. Полосы определенного вида, например, полоса **rbDetail**, при

формировании отчета создаются автоматически для каждой записи набора данных.

В процессе выполнения приложения при вызове метода **Preview** также генерируются события **BeforePrint** и **AfterPrint**.

После создания полосы определенного типа в ней размещаются соответствующие компоненты, при этом необходимо использовать только **компоненты страницы QReport**. Размещение на полосе других компонентов, например, **Label** или **Edit**, не вызывает ошибки трансляции, но в сформированный отчет такие компоненты не попадают.

Можно разместить компонент отчета и вне полосы — непосредственно на компоненте-отчете **QuickRep**. В этом случае он будет выводиться на каждой странице отчета (например, компонент **QRImage**, содержащий логотип фирмы).

Обычно используются следующие компоненты отчета:

- **QLabel** — надпись, содержащая текст (аналог надписи **Label**), которая может размещаться на любой полосе, но для полос данных обычно не используется;
- **QRDBText** — значение поля записи (аналог компонента **DBText**); обычно размещается в полосе данных;
- **QRExpr** — значение, формируемое на основе выражения, в котором могут использоваться значения полей записей; обычно используется для полос данных и нижних колонтитулов;
- **QRSysData** — системная информация, обычно используемая для итоговых полос и полос колонтитулов; ее вид определяется свойством **Data** типа **TQRSysDataType**, принимающим, к примеру, значения:
 - **qrsColumnNo** — номер текущего столбца (для одноколоночного отчета равен единице);
 - **qrsDateTime** — текущие дата и время;
- **QRImage** — графический образ, загружаемый аналогично графическому образу **Image** (может быть использован в любой полосе, но обычно не размещается в полосах данных); с помощью компонента **QRImage** выводится, например, логотип организации; кроме того, следует отличать этот компонент и от компонента **QRDBImage**, который обычно размещается в полосе данных и отображает рисунок из поля таблицы;
- **QRShape** — геометрическая фигура, размещаемая в любой полосе.

Перед компонентом **QRSysData** может находиться надпись, которая указывается в свойстве **Text** типа **AnsiString**. По умолчанию надпись отсутствует.

Простой отчет

Простой отчет представляет собой отчет на основе данных из одного набора данных и содержит сведения, которые выводятся в табличном виде без дополнительных условий, например, группирования данных. Размещение и вид печатаемых в отчете данных аналогичны размещению и виду данных, ото-

бражаемых в сетке DBGrid. Отличием является то, что данные отчета размещаются не на форме, а на бумажном документе, и их нельзя редактировать.

Рассмотрим действия, выполняемые при подготовке простого отчета, на примере перечня книг по программированию (см. рис. 11.23). Многие из этих действий используются и при разработке отчетов других видов. Для создания простого отчета требуется выполнить следующее:

- разместить на форме компонент QuickRep;
- создать для компонента QuickRep требуемые полосы отчета;
- разместить в полосах нужные компоненты отчета, чаще всего QRLabel, QRDBText и QRExpr;
- создать для событий, например, нажатия кнопок с заголовками **Печать** и **Просмотр**, обработчики, в которых вызываются методы печати и предварительного просмотра отчета соответственно.

Размещение на форме компонента отчета, установка параметров отчета, а также создание обработчика события нажатия кнопки печати рассмотрено при описании компонента QuickRep выше.

Простой отчет может содержать следующие полосы, перечисляемые в порядке их размещения на странице:

- верхний колонтитул (rbPageHeader);
- заголовок отчета (rbTitle);
- заголовки столбцов (rbColumnHeader);
- данные (rbDetail);
- итог отчета (rbSummary);
- нижний колонтитул (rbPageFooter).

На этапе разработки название каждой полосы выводится серым цветом в ее левом нижнем углу.

Можно добавить к проекту шаблон простого отчета, вызвав командой меню File/New/Other... (Файл/Новый/Другой) Хранилище объектов и выбрав на странице Forms (Формы) объект QuickReport List (Лист отчета). Шаблон этого отчета содержит полосы заголовка отчета, заголовков столбцов и данных, нижний колонтитул и расположен на отдельной форме QRListForm. На форме отчета также расположен набор данных Table.

Заголовок отчета выводится один раз на первой странице сразу под верхним колонтитулом, если он есть. В полосе заголовка обычно размещаются надписи QRLabel, содержащие требуемый текст (как правило, в качестве заголовка выводится название всего отчета). При необходимости в заголовке можно разместить, например, сведения о названии, адресе и телефоне организации, а также логотип. В приведенном на рис. 11.23 примере в полосе заголовка находится название отчета. Для вывода названия отчета используется компонент QRLabel, в котором набран текст Книги по программированию.

Полосы заголовков столбцов и данных являются основными полосами, в которых размещаются компоненты, обеспечивающие табличный вывод содержимого набора данных. Заголовки столбцов выводятся на каждом листе отчета.

Для заголовков столбцов данных в полосу заголовков обычно помещаются компоненты **QRLabel**, в которые заносится текст, соответствующий полям данных.

Для вывода значений полей записей в полосу данных обычно помещаются компоненты **QRDBText** и **QRExpr**. Более простым является использование компонентов **QRDBText**, каждый из которых отображает значение связанного с ним поля. Имя набора данных указывается в свойстве **DataSet**, а имя поля задается в свойстве **DataField**. Для наглядности схематично состав простого отчета и его связи с набором данных приведены на рис. 11.24.

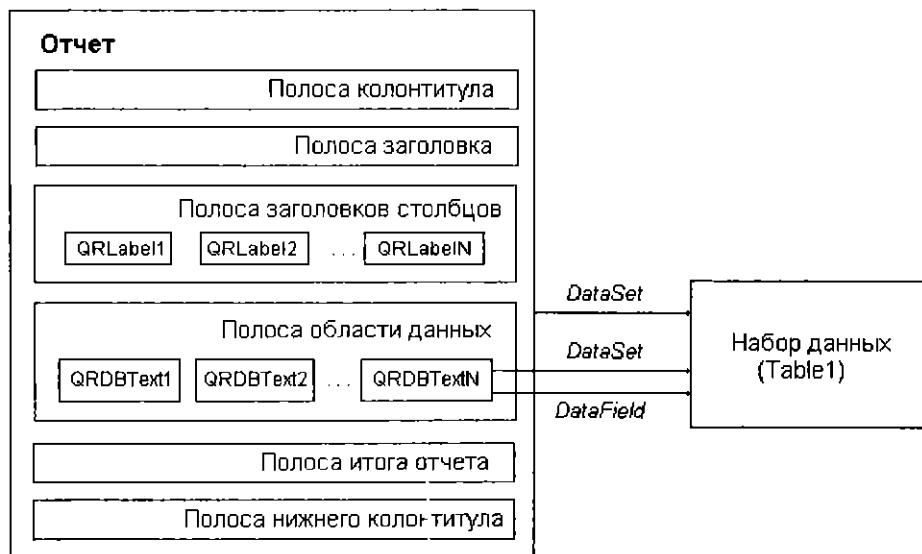


Рис. 11.24. Состав простого отчета и его связи с набором данных

На этапе разработки в отчете присутствует только одна полоса данных, но при формировании отчета отдельная полоса данных будет выведена для каждой записи отчета. Напомним, что если набор данных является пустым и не содержит записей, то область данных не выводится. Чтобы для пустого набора данных были выведены остальные полосы (кроме полос данных), свойству **PrintIfEmpty** компонента **QuickRep** устанавливается значение **true** (по умолчанию).

Компонент **QRExpr** позволяет вставлять в отчет значение выражения, рассчитываемого обычно с участием различных полей записей. Выражение заносится в свойство **Expression** типа **String**, для формирования которого удоб-

но использовать окно **Expression Wizard** (Мастер выражений), вызываемое через Инспектор объектов.

Для вставки в выражение имени поля нужно нажать кнопку **Database Field** (Поле БД) и в открывшемся окне выбрать набор данных и имя поля.

В выражении можно использовать функции, которые разбиты по категориям и выбираются в специальном окне, вызываемом нажатием кнопки **Function** (Функция).

Названия полей и функций можно набирать и вручную, однако это увеличивает вероятность ошибки.

Чтобы протестировать введенное выражение, следует нажать кнопку **Validate**, при этом выполняется проверка выражения, а разработчику выдается сообщение о корректности выражения или об ошибке.

В выражении можно использовать только поля наборов данных, которые размещены на форме отчета. В противном случае набор данных оказывается недоступен для компонента **QRExpr**.

При разработке приложения такие компоненты отчета, как **QRLabel**, **QRDBText** и **QRExpr**, имеют одинаковый внешний вид.

Итоговая полоса отчета выводится один раз в конце отчета сразу после полосы данных. В этой полосе обычно стоят либо итоговые сведения отчета, например, средние и максимальные значения по данным какого-либо поля, либо должность и фамилия лица, утверждающего отчет. В итоговой полосе обычно размещаются компоненты **QRLabel** для вывода надписей и **QRExpr** для вывода значений выражений.

Отметим, в качестве итога может выводиться, к примеру, число записей в таблице или заверительная подпись.

Колонтитулы печатаются в начале и конце каждой страницы, в них обычно выводятся сведения о дате, времени печати, а также номер страницы. Для этого в полосах колонтитулов чаще всего размещаются компоненты **QRSysData**, которым устанавливается требуемое значение свойства **Data**.

В полосе колонтитула можно также разместить и другие компоненты, например, **QRLabel** для вывода на каждой странице названия организации.

Контрольные вопросы и задания

1. Приведите схему взаимосвязей файла проекта при разработке приложения?
2. Какие файлы входят в состав модуля формы и каково их назначение?
3. Охарактеризуйте основные элементы, составляющие интерфейс системы C++ Builder.
4. Каково назначение окон Инспектора объектов и Редактора кода?
5. Какие элементы входят в состав проекта?
6. Что содержит файл представления формы?

7. Каким образом определяется функциональность приложения?
8. Разработайте обработчик события нажатия кнопки.
9. В чем отличие действия команд Run, Make и Build?
10. Охарактеризуйте процесс создания пользовательского интерфейса приложения.
11. Как создается обработчик события для компонента?
12. Какие механизмы доступа к данным поддерживаются в одно- и двухуровневых приложениях, разрабатываемых средствами системы C++ Builder?
13. Охарактеризуйте таблицы, образующие базу данных.
14. Укажите различия в построении индексов в таблицах Paradox и dBase?
15. Охарактеризуйте способы доступа к данным.
16. Что представляет собой связывание таблиц и для чего оно организуется?
17. Изобразите схему связи между таблицами базы данных Paradox.
18. Охарактеризуйте формат таблиц dBase и Paradox.
19. Каково назначение инструментальных средств системы C++ Builder по работе с базами данных.
20. Назовите основные страницы Палитры компонентов, на которых расположены компоненты, используемые при создании приложений для баз данных.
21. Каково назначение компонентов, расположенных на странице QReport?
22. Какие классы исключений используются для операций, связанных с работой приложений для баз данных?
23. Какие события для обработки исключений имеются у набора данных Table?
24. Каково назначение и возможности программы Database Desktop?
25. Какие действия выполняются при определении структуры таблицы?
26. Выполните создание таблицы с помощью программы Database Desktop с назначением полей 5 различных типов.
27. Каким образом назначается тип поля в таблице?
28. Каким образом выполняется задание индекса в таблице?
29. Выполните изменение индекса в имеющейся таблице.
30. Назовите возможные варианты ограничений на значения полей.
31. Поясните, в чем проявляется понятие ссылочной целостности.
32. Каким образом можно выполнить изменение структуры таблицы?
33. Назовите компоненты, используемые при создании приложения для базы данных.
34. Для чего предназначен компонент DataSource?
35. Какую роль в приложении играет компонент DBGrid?
36. Какие действия в приложении для базы данных позволяет выполнять компонент DBNavigator?

37. Изобразите схему, поясняющую взаимосвязь компонентов приложения для базы данных и таблицы.
38. Какие компоненты используются при подготовке отчета?
39. Осуществляется связывание отчета с набором данных?
40. Какие полосы могут входить в состав отчета?
41. Какие события генерируются при печати отчета?
42. Укажите действия, выполняемые при создании простого отчета.
43. Изобразите схему, поясняющую состав простого отчета и его взаимосвязи с набором данных.

Литература

1. Программирование на С++: Учебное пособие /В. П. Аверкин, А. И. Бобровский, В. В. Веснич, В. Ф. Радушинский, А. Д. Хомоненко; Под ред. проф. А. Д. Хомоненко. — СПб.: КОРОНА прнт, 2003.
2. Архангельский А. Я. Программирование в С++ Builder 6. — М.: ЗАО «Издательство БИНОМ», 2002.
3. Хомоненко А. Д., Агадуров С. Е. Работа с базами данных в С++ Builder. — СПб.: БХВ-Петербург, 2006.

12. СУБД Visual FoxPro 8.0

Visual FoxPro 8.0 представляет собой СУБД реляционного типа с развитыми средствами создания БД, организации запросов к ним, построения приложений с использованием визуального, объектно-ориентированного программирования. СУБД Visual FoxPro 8.0 может работать в среде Windows 98/ME/XP/2000.

12.1. Общая характеристика

В Visual FoxPro реализованы все атрибуты реляционной СУБД. Так, в ней введено понятие *базы данных* как совокупности связанных таблиц, информация о которых хранится в *словаре данных*. В БД определяются условия ее целостности с помощью первичных и внешних ключей таблиц. Все события (изменения), происходящие в БД, могут обнаруживаться и централизованно обрабатываться в связи с появлением в Visual FoxPro таких средств, как *триггеры и встроенные процедуры*.

Visual FoxPro совместима с более ранними версиями FoxPro. Поэтому приложения, разработанные в более ранних версиях FoxPro, могут быть адаптированы с ее помощью в среду Windows. Visual FoxPro обладает высокой скоростью в обслуживании БД.

Используя стандарт ODBC и SQL-запросы для выборки данных Visual FoxPro позволяет работать с БД СУБД Access, Paradox, dBase и т. д., с серверами БД — Microsoft SQL Server, Oracle и др.

Приложение Visual FoxPro может одновременно работать как с собственными, так и сетевыми таблицами, расположенными на других компьютерах локальной сети.

Поддерживаются такие механизмы работы с Windows приложениями как:

- Dynamic Data Exchange (DDE) — динамический обмен данными;
- Object Linking and Embedding (OLE) — встраивание и связывание объектов.

При работе Visual FoxPro 8.0 к аппаратным ресурсам предъявляются следующие требования:

- IBM-совместимый компьютер с процессором Pentium 133 и выше;
- мышь;
- основная память объемом 64 Мбайта, рекомендуется объем 128 Мбайта или выше;

- 165 Мбайтов дискового пространства;
- монитор VGA или с более высоким разрешением.

С помощью Visual FoxPro можно создавать сетевые приложения, функционирующие в сетях под управлением Microsoft Windows 9x/2000, Microsoft LAN Manager и др.

Для упрощения процесса разработки баз данных и приложений имеется большое количество Мастеров, Конструкторов и Построителей.

Некоторые количественные характеристики базы данных Visual FoxPro: максимальное количество записей – 1 000 000 000; максимальный размер таблицы – 2GB; максимальное количество символов в записи – 65 500; максимальное количество полей в записи – 255; максимальное количество одновременно открытых таблиц – 2551.

12.2. Новые возможности Visual FoxPro 8.0

В Visual FoxPro 8.0 по сравнению с предыдущей версией продукта появились новые возможности, перечисленные ниже.

- Усовершенствован интерфейс интегрированной среды разработки и отладки приложений.
- Введены новые команды, базовые классы и функции.
- Появились новые возможности по работе с документами XML и Web-службами.
- Усовершенствована совместимость продукта с системой Microsoft Visual Studio .NET и сервером баз данных Microsoft Server 2000.
- Введен Task Pane Manager (Менеджер панели задач), который позволяет улучшить разработку приложений, упростить запуск Мастеров создания приложений, запуск примеров, открытие приложений, баз данных и справочной системы и т. п.
- Расширены возможности Конструктора таблиц.
- Расширены возможности Конструктора меню. В частности, при разработке меню допускается перемещать пункты меню по его иерархической структуре.
- Добавлен новый класс **CursorAdapter**, с помощью которого реализуются функции универсального доступа к данным.
- В редакторе кода появились новые возможности расширенного поиска информации, автоматического форматирования текста и цветной печати исходного кода приложения.

12.3. Элементы проекта

При создании проектов, баз данных, таблиц, запросов, форм, отчетов, приложений и других элементов в среде Visual FoxPro для каждого из названных элементов формируется отдельный файл. При этом имя файла любого элемента пользователь может задать любое, расширение имени файла формируется автоматически и помогает в идентификации этих элементов (объектов). Перечень элементов проектов Visual FoxPro и соответствующих им расширений имен файлов приведен в табл. 12.1. Файлы элементов, созданных на базе других (родительских) элементов, имеют общие с ними имена. К примеру, Memo-поле и поле типа General создаются на базе таблицы БД, поэтому имена их файлов совпадают с именами соответствующих таблиц, а расширения указывают на назначение этих файлов.

Таблица 12.1
Элементы проектов Visual FoxPro и расширения имен файлов

Компоненты	Расширения имен, примечания
Приложение	APP, сгенерированная программа EXE, выполнимая программа
Проект	PJX PJT, Memo-поле
База данных	DBC DCT, Memo-поле DCX, индекс
Таблица Visual FoxPro	DBF, FPT, Memo-поле
Одиночный индексный файл	IDX
Составной индексный файл	CDX
Мемо-поле и поле типа General	FRT.
Форма	SCX, SCT, Memo-поле
Запрос	QPR, сгенерированная исходная программа QPX, программа после компиляции
Отчет	FRX, FRT, Memo-поле
Этикетка	LBX, LBT, Memo-поле

Таблица 12.1 (продолжение)

Компоненты	Расширения имен, примечания
Меню	MNX, описание облика меню MNT, Memo-поле MPR, сгенерированная исходная программа MPX, программа после компиляции
Библиотеки	VCX, класса VCT, Memo-поле библиотеки класса DLL, динамических связей Windows FLL, динамических связей Visual FoxPro
Программа	PRG, исходный текст FXP, после компиляции
Ошибки компиляции	ERR
Файл формата	FMT
Описание окружения	VUE
Рисунок	BMP
Звуковая запись	WAV
Текст TXT	
Экран (предыдущие версии FoxPro)	SPR, сгенерированная исходная программа SPX, программа после компиляции

Раскроем кратко назначение указанных элементов Visual FoxPro.

Проект является основным средством объединения отдельных элементов Visual FoxPro и управления ими. С помощью проекта Visual FoxPro осуществляет поиск и собирает вместе файлы проекта, отслеживает текущие версии элементов, перекомпилирует программы, обновляет экранные формы, меню и т. д. Из проекта осуществляется генерация приложения (APP-файл) или исполняемого приложения (EXE-файл). Вся информация о проекте хранится в специальной таблице – файле с расширением PJX и соответствующем Мето-файле с расширением PJT.

База данных представляет собой совокупность связанных таблиц, а также включает словарь БД, триггеры и процедуры обработки событий.

Словарь БД хранит описание структуры БД и представляет собой совокупность системных таблиц.

Триггеры срабатывают при определенных изменениях (событиях), происходящих в БД, и вызывают для обработки процедуры, принадлежащие БД.

Создание БД осуществляется с использованием Конструктора БД, который позволяет создавать, индексировать, модифицировать и связывать таблицы БД между собой. Вся информация о БД хранится в файле с расширени-

ем DBC. Для поддержки предыдущих версий FoxPro возможно создание отдельных таблиц, не объединенных в БД. Каждая отдельная или входящая в БД таблица размещается в файле с расширение DBF.

Индексы. Таблицы могут быть проиндексированы (упорядочены) по некоторым полям, а значит иметь связанные с ней один или несколько индексных файлов, хранящих необходимую для индексации информацию. Индексы используются также для ускорения поиска информации в таблицах.

Мемо-поля содержат текстовую информацию, поля типа General служат для организации обмена данными с другими приложениями Windows.

Формы используются для ввода и редактирования данных в таблицах. Формы предоставляют пользователю удобный интерфейс для доступа к хранимым данным с возможностью отображения их в требуемом виде. С помощью Конструктора форм можно создавать формы любой степени сложности.

Запросы являются средством извлечения информации из БД, которая может содержаться в нескольких ее таблицах. При этом можно использовать Конструктор запросов или создавать запрос вручную с использованием SQL-команд Visual FoxPro. Конструктор запросов позволяет создавать запросы по образцу. При этом обеспечивается возможность на основе визуальной информации о таблицах извлечь нужные данные и представить их в удобном виде (в виде таблиц, графиков или диаграмм). К программированию запросов с помощью SQL-команд прибегают в случаях, когда возможностей Конструктора запросов недостаточно.

Отчеты используются для отображения информации, содержащейся в БД, и позволяют осуществлять в нем необходимую группировку данных, отображать итоговые и расчетные данные. Они могут создаваться вручную и с помощью Конструктора отчетов.

Этикетки представляют собой почтовые адреса рассылки, нечтаемые на конвертах. По сути этикетки являются мини-отчетами, нечтаемыми на конвертах с выбором информации из соответствующей таблицы. Каждая запись таблицы содержит адрес одной из фирм-клиентов. Для создания этикеток может использоваться Мастер этикеток.

Меню является основной частью приложения (если таковое создается), управляющей работой приложения и его компонентов.

В процессе создания меню с помощью Конструктора меню можно выделить следующие три этапа: конструирование, генерация и компиляция.

На этапе конструирования меню создается таблица, хранящаяся в файле с расширением MNX. Этой таблицей можно манипулировать как и любой таблицей Visual FoxPro, а отличное расширение сделано для идентификации ее содержимого. Этот файл обычно добавляется в проект.

На втором этапе из названной таблицы генерируется обычный для Visual FoxPro код. Файл с этим кодом имеет то же имя, что и файл MNX (если пользователь не укажет другое), но расширение MPR.

На третьем этапе после компиляции из MPR-файла программы меню появляется в файле с расширением MPX.

Библиотеки классов предназначены для хранения классов, созданных в Visual FoxPro. Классы служат для описания объектов, используемых в объектно-ориентированном программировании (ООП). Библиотеки классов наиболее часто применяются для создания экранных форм.

Программы, написанные на языке Visual FoxPro, реализуют различные функции в приложении. Файлы с программами являются текстовыми, формируются с помощью встроенного редактора и имеют расширение PRG.

Файлы описания окружения с расширением VUE хранят информацию об открытых таблицах, активных индексах и установленных между таблицами связях.

Рисунки и звукозаписи соответственно хранятся в файлах с расширениями BMP и WAF или в полях типа General таблиц. Этой информацией Visual FoxPro-приложение может обмениваться с другими Windows-приложениями.

12.4. Интерфейс Visual FoxPro

Работа по созданию баз данных и приложений выполняется в главном окне Visual FoxPro (рис. 12.1). Состав элементов в главном окне может настраиваться пользователем в процессе работы.

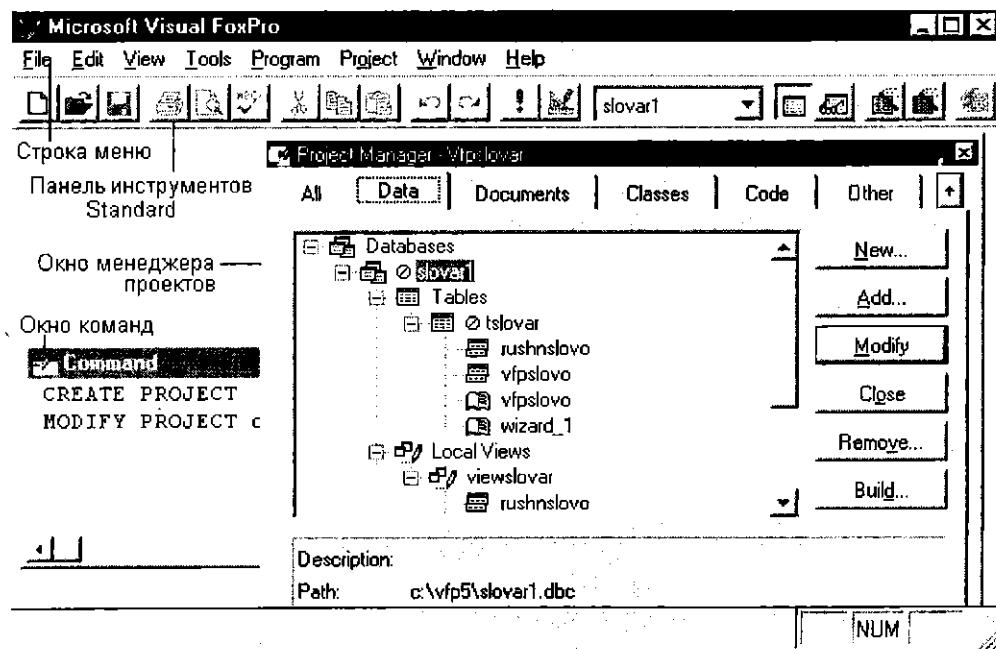


Рис. 12.1. Главное окно Visual FoxPro

Обязательным элементом главного Visual FoxPro является строка меню. С ее помощью можно задавать команды настройки состава окна, вызывать средства автоматизации (Мастера, Конструкторы и Построители).

В основном окне Visual FoxPro обычно отображается панель инструментов Standard (стандартная). С помощью кнопок этой панели удобно задавать наиболее часто используемые команды по работе с файлами элементов Visual FoxPro: New (Создать), Open (Открыть), Save (Сохранить), Print One Copy (Печатать один экземпляр); команды работы с выделенным фрагментом с помощью буфера обмена: Cut (Вырезать), Copy (Копировать), Paste (Вставить) и ряд других.

В основном окне Visual FoxPro (рис. 12.1) можно установить отображение следующих панелей инструментов: Color Palette (Цветовая палитра), Database Designer (Конструктор баз данных), Form Controls (Элементы управления форм), Form Designer (Конструктор форм), Layout (Размещение элементов управления в отчете или форме), Print Preview (Предварительный просмотр), Query Designer (Конструктор запросов), Report Controls (элементы управления отчетов), Report Designer (Конструктор отчетов) View Designer (Конструктор просмотров). Изменение состава панелей инструментов выполняется с помощью команды View | Toolbars....

При работе с любым элементом проекта (базой данных, таблицей, запросом и др.) для задания команд удобно пользоваться соответствующим контекстным (всплывающим) меню, вызов которого выполняется щелчком правой кнопкой мыши.

Окно команд (при условии его отображения в основном окне Visual FoxPro) служит для ввода команд SQL. Кроме того, в случае выполнения каких-либо действий над базой данных в окне команд отображаются соответствующие им команды SQL. Для задания отображения окна команд служит команда Window | Command Window (окно | командное окно). Отмену отображения окна команд можно выполнить щелчком мыши по кнопке в правом верхнем углу окна.

Окно Project Manager (Менеджер проектов) отображается при создании нового или при открытии существующего проекта. Создание базы данных, запросов, отчетов и ряда других элементов может выполняться и без создания проекта. Однако использование проекта создает определенные удобства в работе с различными элементами Visual FoxPro, входящими в состав одного проекта.

12.5. Средства автоматизации разработки

Для автоматизации разработки баз данных и приложений в среде Visual FoxPro имеются следующие средства визуального программирования: Мастера (Wizards), Конструкторы (Designers) и Построители (Builders).

Мастера позволяют сконструировать требуемый объект, например, таблицу, просмотр или метку, путем выбора одного из предлагаемых вариантов оформления объекта в ходе пошаговой (с ограниченным числом шагов) процедуры формирования объекта.

Конструкторы предоставляют существенно больше возможностей по формированию облика создаваемого объекта. Естественно, что при этом может потребоваться несколько больше времени.

Построители помогают формировать отдельные элементы управления при создании объекта или комбинировать конструкции при создании выражений. Можно сказать, что Построители являются аналогами Конструкторов, ориентированными на автоматизацию создания составляющих элементов объектов.

Мастера удобно использовать при необходимости создать в кратчайший срок несложное приложение. Если возможностей Мастеров окажется недостаточно, целесообразно прибегнуть к помощи Конструкторов и Построителей. При создании крупных проектов могут потребоваться объектно-ориентированные средства Visual FoxPro создания пользовательских классов, упрощающие создание пользовательских интерфейсов программ проекта.

12.6. Создание баз данных

Прежде чем рассматривать процедуры создания БД и таблиц познакомимся с типами данных, используемых в них.

Перечень типов данных и краткая их характеристика приведены в таблице 12.2.

Таблица 12.2
Типы данных Visual FoxPro

Обозначение	Тип	Диапазон	Объем памяти, байт	Описание
A	Array			Массив данных некоторого типа
B	Double	от +/-4.94065648541247E-324 до +/-1.79769313486232E+308.	8	Число с плавающей точкой двойной точности
C	Character	Любые символы	1–254	Текстовая (символьная) строка
D	Date	от 01/01/100 до 12/31/9999	8	Дата
F	Float	от -0.9999999999*10+19 до 0.9999999999*10+20.	8	Такое же как Numeric
G	General	Определяется доступной памятью	4 (в DBF)	Ссылка на OLE объект

Таблица 12.2 (продолжение)

Обозначение	Тип	Диапазон	Объем памяти, байт	Описание
I	Integer	- 2147483647 до 2147483646	4	Число целое
L	Logical		1	Логическое значение
M	Memo	Определяется доступной памятью	4 (в DBF)	Ссылка на примечание
N	Numeric	от -0.99999999999*10+19 до 0.9999999999*10+20.	8	Число с фиксированной точкой целое или дробное; допускает от 1 до 20 символов в таблице
T	DateTime	от 01/01/100 до 12/31/9999 и от 00:00:00 утра до 23:59:59 вечера	8	Дата и время
Y	Currency	от -22337203685477.5807 до 922337203685477.5807.	8	Денежное значение

Приведенные в первой колонке таблицы буквенные обозначения используются для отображения типа используемых переменных. Просмотреть список переменных можно по команде DISPLAY MEMORY.

Данные каждого типа могут храниться в полях таблиц этого же типа. Заметим, что Visual FoxPro не имеет команд определения типов переменных. Определение типов выполняется при присваивании переменным первоначальных значений. Рассмотрим подробнее перечисленные типы данных.

Character

Текстовый (символьный) тип используется для побайтного хранения символьных строк длиной от 1 до 254 символов. Элементами строк могут быть печатные знаки — буквы, цифры, пробелы и знаки препинания. Константа символьного типа должна быть заключена в разделители, например:

“строка” ‘строка’ [строка]

Возможно сравнение символьных строк в соответствии с алфавитным порядком, например:

“ < “а”=.T. “арба”<“арбуз”=.T.

Здесь .T. — логические значение (true). Сравнение идет до первого несовпадающего символа, или до окончания правой строки. Операторы сравнения те же, что и в NUMERIC. Дополнительно введена операция точного сравнения (длин строк и всех символов), обозначаемая как ==. Сравниваемые сим-

волы должны быть набраны в одинаковом (нижнем или верхнем) регистре. Имеются два оператора склеивания строк + и - .

Пример:

“Весна” + [96] = [Весна 96] или

“Весна” - [96] = [Весна96]

В Visual FoxPro добавлен новый тип символьных полей Character (binary), которые позволяют хранить символы с ASCII-кодами от 0 до 255.

Следующие 4 типа данных (Numeric, Float, Integer и Double) можно условно объединить в одну общую группу числовых данных. Допустимыми символами полей перечисленных типов являются цифры. При работе с данными этих типов возможно выполнение математических операций и автоматический контроль вводимых данных при работе с Visual FoxPro, исключающий ввод любых символов, кроме цифр.

Numeric

Числа в формате с фиксированной точкой. Целое отличается отсутствием дробной части. Например, оператор присваивания

x=43.385

определяет тип переменной x как NUMERIC и присваивает ей указанное значение. Над данными этого типа допустимы два типа операций:

1. Арифметические

+ , - , * , / ,

** или ^ — возведение в степень.

2. Сравнения

< , > , = , <= , >=,

или <> или != — не равно.

Float

Числовые данные типа Float включены для совместимости и функционально эквивалентны данным типа Numeric.

В Visual FoxPro появились новые типы числовых данных Double и Integer.

Double

Числовые вещественные (с плавающей точкой) данные двойной точности.

Integer

Данные целочисленного типа применяются для представления целых чисел и позволяют сэкономить место для хранения данных.

Date

Этот тип используется для хранения календарных дат. При этом применяются различные форматы представления данных, например:

AMERICAN 12/31/93

GERMAN 31/12/93 и др.

Формат AMERICAN используется по умолчанию. Иной вид формата устанавливается командой

SET DATE <формат>,

где <формат> — вид заказываемого формата. Поддерживается контроль правильности вводимых дат. Даты изменяются в диапазоне 01.01.100 до 12.31.9999. При задании дат ХХ века можно указывать только две последние цифры года.

DateTime

Тип *дата и время* появился в Visual FoxPro. Кроме 8 байт, которые требовались для хранения дат в формате Date, под данные данного типа отводится дополнительно еще 6 байтов для хранения времени в виде HHMMSS, где HH — часы от 00 до 23, MM — минуты, а SS — секунды. При превращении данных типа Date в данные типа DateTime автоматически устанавливается время 12:00:00. Над данными этого типа можно выполнять определенные арифметические операции, так, например, добавляя к дате 1 мы увеличиваем ее на один день, а добавлять секунды к переменной типа DateTime мы изменяем соответственно время.

Logical

Логический тип данных допускает два возможных значения и четыре варианта их обозначения:

Истина (Да) — .T. или .t. или .Y. или .у.

Ложь (Нет) — .F. или .f. или .N. или .п.

Результат сравнения данных любого типа является логическим значением:

.T. — если условие сравнения соблюдается,

.F. — в противном случае.

Над данными логического типа могут выполняться следующие операторы:

.NOT. — НЕ (отрицание или невыполнение условия);

.AND. — И (одновременное выполнение двух условий);

.OR. — ИЛИ (выполнение хотя бы одного условия).

Memo

Поля базы данных (БД) данного типа предназначены для хранения символьных строк произвольной длины. Значения типа Memo могут иметь произвольный размер, определяемый размером только жесткого диска компьютера и хранятся в отдельном файле с расширением .FPT, имя которого совпадает с именем соответствующей таблицы. Каждая таблица имеет толь-

ко один Мемо-файл вне зависимости от того, сколько Мемо-полей она имеет. В Мемо-поля DBF-файлов заносятся лишь ссылки (указатели) на соответствующие символьные строки. Значение Мемо-поля можно присвоить переменной символьного типа и далее работать с ней как с символьной константой.

Само Мемо-поле имеет блочную структуру. Размеры блока изменяются с помощью команды SET BLOCKSIZE. При этом могут задаваться блоки в диапазоне от 33 до 511 байт. Размер блока большего размера кратен 512 байтам и обозначается целыми числами от 1 до 32. По умолчанию устанавливается блок размером в 64 байта. При этом запись в 65 байт потребует 2 блока, под которые будет отведена область памяти в 128 байт. При выборе размеров блоков Мемо-полей необходимо стремиться к балансу между перерасходом памяти, если блоки чрезмерно велики, и снижением скорости работы с ними из-за увеличения блоков, обусловленного их небольшими размерами.

Замечание.

FPT-файл, созданный для какой-то таблицы, является неотъемлемой ее частью.

General

Поля баз данного типа позволяют хранить двоичные данные, а именно изображения, звук и т. д. Поля типа General являются специальной разновидностью Мемо-полей. Они хранятся в том же FPT-файле, что и все Мемо- поля данной таблицы, однако используются они иначе, чем обычные Мемо- поля.

Currency

Это тип данных введен в Visual FoxPro для оперирования денежными суммами. Поля таблиц данного типа схожи с числовыми полями, но в отличие от числового поля, для них определена точность в четыре знака после запятой. Поэтому при отображении целых чисел после денежной точки добавляются 4 нуля. Например, не 36, а 36.0000.

Array

Массив представляет собой совокупность элементов, имеющих общее имя. Массив объявляется командой *DECLARE* или *DIMENSION*, которая задает имя и длину массива в круглых скобках, например:

DECLARE mas1(4,6), mas2(10)

Здесь описаны два массива: двумерный массив с именем mas1 (размерность 4 на 6) и одномерный массив из 10 элементов.

При объявлении массива каждому его элементу по умолчанию присваивается начальное значение .F. (*false*). Каждый элемент массива может использо-

ваться как переменная любого допустимого типа и размера. Поэтому с ними можно и работать как с переменными: присваивать значения, включать в вычисляемые выражения, распечатывать значения и т. д.

Обращаются к элементу массива по имени массива с указанием индекса — номера этого элемента в массиве, например, *mas(i)*.

Создание БД с помощью Конструктора

Конструктор БД позволяет создавать, модифицировать и индексировать таблицы БД, устанавливать постоянные межтабличные связи.

Создание базы данных с помощью Конструктора БД выполним применительно к базе данных «Учебная», спроектированной в предыдущем разделе. Прежде всего, определим типы и размеры полей БД следующим образом:

```
T_ZAGR(Fio, Predm, Gruppa, VidZan),
C15 C10 I3 C1
T_PREP(Fio, Stag, Dolgn, Kaf),
C15 I2 C10 I2
T_DOLGN(Dolgn, Oklad),
C10 N7
T_STAG(Stag, D_Stag)
C12 N7
```

В рассматриваемой БД подчеркнутые поля образуют ключ соответствующей таблицы. В таблице T_ZAGR поля (Fio, Predm, Gruppa) образуют ключ при условии, что один и тот же преподаватель в одной группе не может одновременно читать лекции и проводить практические занятия. В противном случае ключ таблицы будет включать все ее поля. В таблице T_PREDM ключ образуют поля Predm и Gruppa с учетом допущения, что один и тот же предмет может читаться в различных группах.

Замечание.

Наименования полей в рассматриваемой БД по традиции (в связи с ограничениями FoxPro) заданы латинскими буквами (Fio, Predm, Gruppa, Stag и т. д.), хотя в Visual FoxPro 5.0 полям таблицы допускается задавать названия, отличные от латинских.

Перейдем непосредственно к созданию БД «Учебная». Для удобства назовем ее Bdu.

Для создания БД выполним команду **File | New (Файл | Создать)**. В результате откроется диалоговое окно **New (Новый)** (рис. 12.2), в котором перечислены все типы файлов Visual FoxPro, которые можно создать.

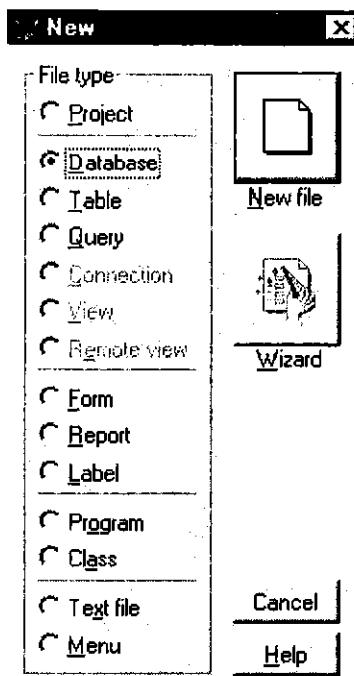


Рис. 12.2. Диалоговое окно New

При выбранном переключателе **Database** (база данных) нажатие кнопки **New file** (Новый файл) приводит к появлению диалогового окна **Create** (Создать). В списке верхней части окна следует выбрать папку, в поле **Enter** (ввод) указать имя файла базы данных (**Bdu.dbc**) и нажать кнопку **Сохранить**.

В результате откроется диалоговое окно **Database Designer – Bdu** (окно Конструктора созданной базы данных **Bdu**). В системном меню **Visual FoxPro** появится новое меню **Database** (база данных), содержащее команды Конструктора БД (табл. 12.3).

Таблица 12.3

Команды меню Database

Команда	Назначение
New Table	Создание таблицы
Add Table	Добавление таблицы в БД
New Remote View	Создание удаленного представления данных
New Local View	Создание локального представления данных

Таблица 12.3 (продолжение)

Команда	Назначение
Modify	Модификация таблицы
Browse	Редактирование таблицы в режиме Browse
Remove	Удаление таблицы из БД
Rebuild Table Indexes	Перестройка индексов
Remove Deleted Records	Физическое удаление помеченных записей
Edit Relationship	Редактирование межтабличных связей
Referential Integrity	Определение условий целостности данных
Edit Stored Procedures	Редактирование хранимых процедур
Clean Up Database	Очистка БД от помеченных на удаление объектов

Создать БД можно также, набрав в командном окне Visual FoxPro команду

CREATE DATABASE ИмяБазыДанных

Если имя БД в команде не указано, то при ее выполнении открывается диалоговое окно **Create (создать)**, в котором нужно указать имя создаваемой БД.

12.7. Таблицы и индексы

В Visual FoxPro можно создавать таблицы, входящие в БД, и отдельные таблицы. Рассмотрим создание таблиц, являющихся компонентами БД.

Создание таблиц

При создании таблицы ей присваивается имя, отражающее существование хранимой информации. При присвоении имен таблицам необходимо придерживаться следующих правил:

- каждая таблица в БД должна иметь уникальное имя;
- имя таблицы является именем одноименного файла, возможное число символов в имени определяется операционной системой;
- имя таблицы может содержать буквы, цифры и знаки подчеркивания.

Для создания входящих в БД таблицы требуется выполнить команду **Database | New Table (база данных| новая таблица)**. Далее в диалоговом окне **New Table (новая таблица)** следует вызвать Конструктор таблиц (кнопка **New Table – новая таблица**) или Мастер таблиц (Table Wizard).

Пример 1. Рассмотрим создание для нашей БД таблицы Т_ZAGR(Fio, Predm, Gruppa, VidZan) с помощью Конструктора таблиц.

1. Перейдем в Конструктор таблиц, нажав в окне диалога **New Table** (новая таблица) кнопку **New Table** (новая таблица). Определим имя создаваемой таблицы в окне **Create** (создать). В результате откроется диалоговое окно Конструктора таблиц **Table Designer** (рис. 12.3), позволяющее сформировать таблицу.
2. Выберем вкладку **Fields** (поля) (рис. 12.3), введем имя первого поля Fio. Для задания типа поля перейдем в следующий столбец с помощью мыши или клавиши Tab. Тип поля в нем выбирается из раскрывающегося списка.
3. В следующем столбце установим размер поля. Для поля Fio это 15 символов.
4. Установим заголовок поля. Для этого перейдем в поле **Caption** (заголовок), расположенное в области **Display** и введем там заголовок «Фамилия И.О.». В поле **Field Comment** (комментарий поля) можно задать краткое описание поля.

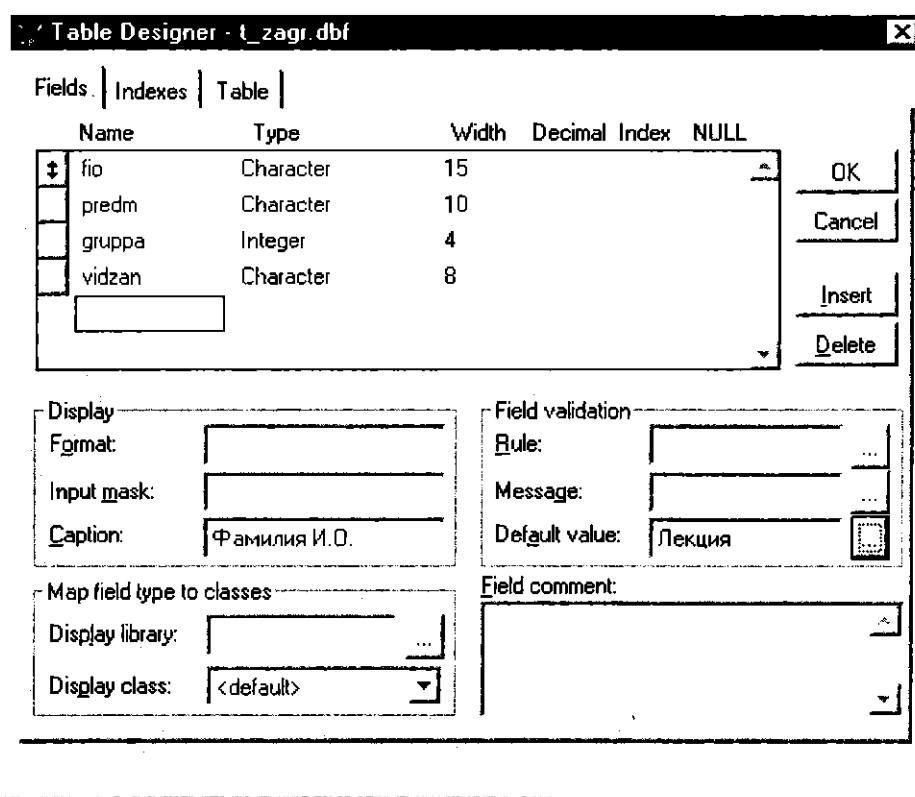


Рис. 12.3. Диалоговое окно Table Designer

5. Для задания сортировки данных в таблице по полю Fio надо перейти в столбец Index (индекс) и из раскрывающегося списка выбрать Ascending (возрастающий), если требуется сортировка в порядке возрастания данных, и Descending (убывающий) – по убыванию. Эта сортировка достигается использованием индексирования, о котором поговорим позже.
6. Аналогично определим остальные поля таблицы. При необходимости изменения порядка следования полей таблицы воспользуемся крайней левой кнопкой вкладки диалогового окна.
7. Полю VidZan (вид занятий) таблицы определим значение по умолчанию. Для этого выберем поле VidZan таблицы и в нижней части окна в поле Default value: (значение по умолчанию:) введем слово «Лекция».

Укажем назначение важнейших элементов диалогового окна.

Decimal – задает число цифр после десятичной точки в соответствующих типах данных,

Index – позволяет указать поле таблицы по которому она будет проиндексирована,

NULL – задает признак поля, позволяющий оставлять это поле пустым при вводе данных. Пустое поле в зависимости от его типа может быть пустой строкой, числом ноль или false (логическая ложь).

Назначение некоторых кнопок:

Insert (вставка) – вставка поля перед тем, на который установлен курсор,
Delete (удаление) – удаление выбранного поля.

Для формирования свойств таблиц необходимо перейти на вкладку Table (таблица) диалогового окна Конструктора таблиц. На этой вкладке можно ввести текстовый комментарий к таблице (поле Table Comment:), задать условия проверки вводимых данных на уровне записей (список Rule:) и задать триггеры (Triggers) – специальные подпрограммы, срабатывающие при выполнении таких операций с записями таблицы как добавление, удаление и изменение. При этом контроль вводимых данных будет осуществляться при каждом обращении к таблице.

Индексирование таблиц

Просматривать и отыскивать данные в таблицах удобнее и быстрее отсортированными (упорядоченными) по одному или нескольким полям. Вводить данные в таблицу упорядоченными хотя бы по одному из полей весьма проблематично. Поэтому данные вводятся в таблицы по мере необходимости и в случайном порядке, а сортировка данных при работе с ними осуществляется с помощью механизма индексирования таблиц.

Под *индексом (индексным выражением)* понимается имя поля таблицы или выражение, включающее совокупность имен полей, по которым упорядочена таблица.

Индексирование таблиц позволяет осуществлять быстрый поиск записей. Механизм индексирования базируется на использовании специального индексного файла, содержащего упорядоченные указатели (ссылки) на записи исходной таблицы, позволяющие извлекать записи в нужном порядке. Поэтому размер индексного файла гораздо меньше размера исходной таблицы.

В Visual FoxPro имеются различные варианты индексных файлов, в том числе и для поддержки предыдущих версий FoxPro. Будем использовать *структурный* индексный файл, который эффективен и прост в применении. Структурный индексный файл обеспечивает реализацию всех индексов одной таблицы и имеет имя, совпадающее с именем самой таблицы, и расширение CDX. Структурные индексные файлы открываются и закрываются одновременно с таблицами, что упрощает работу с ними.

Важным свойством индексов является возможность их использования для организации первичного ключа в таблице. Такие индексы должны быть уникальны, то есть однозначно идентифицировать записи таблицы. Если ключевой индекс состоит из одного поля, то не должно быть двух записей в таблице с одинаковым значением этого поля. Для составного индекса, состоящего из нескольких имен полей, уникальным должна быть вся совокупность значений полей.

В Visual FoxPro тип индексов задается при их создании с помощью Конструктора таблиц. Рассмотрим процедуру создания индекса. В диалоговом окне **Table Designer** (конструктор таблиц) выберем вкладку **Indexes** (индексы) (рис. 12.4).

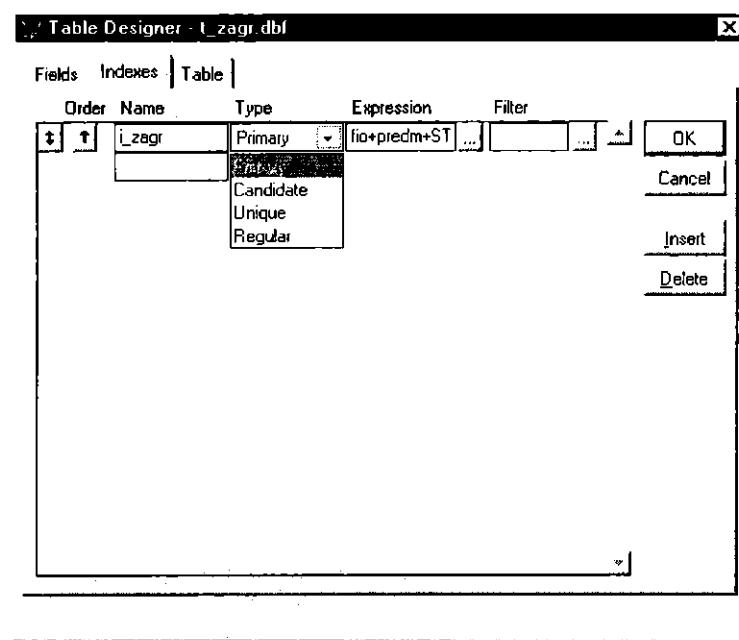


Рис. 12.4 Вкладка Indexes

В поле **Name (имя)** задаем имя тега — имя индекса. Слева от имени индекса располагается переключатель, задающий направление упорядочения значений индексного выражения (по возрастанию или убыванию значений тега).

В поле **Type (тип)** выберем из списка подходящий тип для устанавливаемого тега (табл. 12.4).

Таблица 12.4
Типы индексов (тегов)

Тип тега	Описание
Regular	Значение индексного выражения записывается для каждой записи таблицы. При наличии одного и того же значения для нескольких записей в индексном файле будет указатель на каждую из них. При просмотре таблицы такие записи появляются в порядке их ввода.
Unique	Значение индексного выражения записывается только для первой из повторяющихся записей и только на нее в индексном файле есть указатель. При просмотре таблицы видна только одна (первая) из записей с одинаковым значением индексного выражения.
Candidate	Создается уникальный индекс, не содержащий полей с пустыми значениями. Он является кандидатом на роль первичного ключа, но не является таковым, так как в таблице может быть только один первичный ключ.
Primary	Один из индексов, удовлетворяющий требованиям индекса типа Candidate может быть выбран в качестве первичного (Primary) ключа. Используется для связывания таблиц и определения условий целостности данных.

Поле **Expression (выражение)** позволяет ввести индексное выражение. В простейшем случае индексное выражение может состоять из имени одного поля. В более сложных случаях — это совокупность имен полей или выражение, включающее имена полей, переменные и функции как стандартные, так и пользовательские. Для формирования индексного выражения можно привлечь Конструктор выражений, вызываемый нажатием кнопки справа от поля ввода.

Поле **Filter (фильтр)** позволяет определить для индекса фильтр, служащий для ограничения формируемых значений индекса. Результат выражения, используемого в фильтре должен иметь логический тип.

Пример. Создание индекса, являющегося первичным ключом таблицы.

Рассмотрим формирование первичного ключа для таблицы T_ZAGR. Первичный ключ здесь является составным и представляет собой выражение Fio+Predm+STR(Группа). В подобных выражениях все компоненты должны быть

одного типа. Поэтому следует использовать стандартную функцию STR(), преобразующую числовые значения в символьные. Для создания индекса выполним следующие действия.

1. Откроем окно Конструктора таблиц для таблицы T_ZAGR. Для этого в окне Конструктора БД установим курсор на таблицу, щелчком правой кнопкой мыши вызовем всплывающее меню и выполним команду Modify (модифицировать).
2. В диалоговом окне Конструктора таблиц выберем вкладку Indexes (индексы).
3. В поле Name (имя) открывшегося диалогового окна введем имя индекса i_zagr.
4. В списке возможных типов индекса в поле Type (тип) выберем Primary.
5. В поле Expression (выражение) введем выражение для индекса i_zagr.
6. Установим переключатель Order (порядок) в положение по возрастанию (рис. 12.3).
7. Нажмем OK.

Аналогично можно проиндексировать остальные таблицы БД Bdu (табл. 12.5).

Таблица 12.5
Индексные выражения для индексов БД Bdu

Таблица	Имя тега	Индексное выражение
T_ZAGR	i_zagr	Fio+ Predm+STR(Gruppa)
T_PREP	i_prep	Fio
T_STAG	i_stag	Stag
T_DOLGN	i_dolgn	Dolgn

В остальных таблицах можно использовать простые индексные выражения, состоящие из одного поля.

Изменение структуры таблицы

При работе над проектом порой требуется изменять структуру некоторой таблицы. В структуру таблицы можно вносить следующие изменения:

- изменять имена и типы полей,
- вставлять пропущенные поля,
- удалять лишние поля,
- изменять порядок следования полей в таблице.

Для проведения таких изменений нет особых проблем, и их технологию мы изложим ниже. Проблемы могут возникнуть при выполнении следующих действий:

- переименовании существующих индексных полей, так как это требует перезаписи индексных файлов таблицы;

- изменении длины полей или числа знаков после запятой, так как это может привести к потере данных;
- изменении типа поля, так как в случае невозможности автоматического преобразования данных они будут потеряны.

Замечание.

Перед изменением структуры любой существующей таблицы целесообразно создание резервной копии таблицы и всех ее индексных файлов.

Для изменения структуры таблицы, являющейся элементом БД, нужно открыть диалоговое окно Конструктор таблиц (*Table Designer*) со структурой выбранной таблицы. Для этого достаточно открыть окно Конструктора БД, установить в нем курсор на модифицируемую таблицу и нажать кнопку *Modify Table* (модифицировать таблицу) панели инструментов *Database Designer* (конструктор базы данных).

Рассмотрим подробнее операции по модификации таблицы, выполняемые в окне Конструктор таблиц (*Table Designer*).

Добавление полей является самой безопасной операцией. Для ее реализации необходимо с помощью клавиш-стрелок установить курсор на строку, перед которой необходимо вставить новое поле (в любом ее столбце) и нажать кнопку *Insert* (вставить) (рис. 12.4). При этом появляется новая строка с именем *NewField*. Далее по рассмотренной технологии можно ввести имя нового поля и его параметры.

Для удаления лишнего поля достаточно выделить его в окне Конструктора таблиц и нажать клавишу *Delete* (удалить). Если открытый на данный момент индексный файл не содержит ссылок на удаляемое поле, то поле будет удалено. Если такие ссылки имеются, то появится предупреждающее сообщение.

Переименование полей выполняется следующим образом. Открывается Конструктор таблиц со структурой нужной таблицы, выделяется и редактируется нужное поле, после чего полученная структура сохраняется. Если изменяемое поле входит в индексное выражение, то появляется предупреждающее сообщение. При этом нужно изменить имя поля и отредактировать соответствующие индексные выражения.

Переопределение характеристик полей, а именно типа, длины или числа десятичных знаков в числовом поле, иногда приводит к проблемам. Без проблем выполняется увеличение длин полей. Для этого достаточно в Конструкторе таблиц выделить нужное поле и увеличить его длину. При этом соответствующий DBF-файл будет переписан и приобретет большие размеры.

Проблемы могут возникнуть при попытке уменьшить размеры полей. При этом может произойти усечение строк в символьных полях до нужных размеров. Сокращение целой части числового поля может привести к потере той

части чисел, у которых она не помещается в отведенные размеры (в поле записывается символ звездочки). Сокращение количества знаков в дробной части понижает точность представления числа.

Возможны проблемы и при изменении типов полей. Например, преобразование числового поля в символьное возможно всегда при условии достаточного выделения места для образующихся символьных строк. Это преобразование осуществляется с помощью функции `STR()`. Обратное преобразование выполняется функцией `VAL()` и возможно только в тех случаях, когда строка начинается с цифр или пробелов. В противном случае полученное в результате преобразований числовое поле будет иметь нулевое значение.

Без проблем осуществляется преобразование даты в строку (функция `DTOS()`) и обратное преобразование (функция `CTOD()`), при условии что символьная строка содержит допустимые символы.

12.8. Организация межтабличных связей

В Visual FoxPro можно связывать таблицы — устанавливать связи между ними. При этом возможно установление постоянных и временных связей между таблицами. *Постоянные* связи можно установить в Конструкторе БД, и они прежде всего необходимы для поддержания целостности БД при обновлении содержимого таблиц. Временные связи между таблицами могут устанавливаться при создании экранных форм (для ввода и редактирования данных) и отчетов. Обычно это делается в случаях, когда имеющиеся постоянные связи по каким-либо причинам не подходят для создаваемой формы или отчета.

Постоянныe связи хранятся в словаре БД и автоматически устанавливаются при открытии таблиц. Приведем условия установления постоянных связей.

1. Одна из связываемых таблиц является родительской, другая — дочерней.
2. Для родительской таблицы индекс должен быть определен как первичный ключ (*Primary*) или ключ кандидат (*Candidate*).
3. Для дочерней таблицы индекс может быть не уникальным. По этому индексу дочерняя таблица будет связываться с родительской (внешний ключ).

Создадим постоянные связи между таблицами. Для этого войдем в Конструктор БД и убедимся в наличии (или создадим) перечисленные ниже индексы (табл. 12.6).

Таблица 12.6

Типы индексов БД Bdu

Таблица	Имя тега	Тип индекса	Индексное выражение
T_ZAGR	i_zagr	Primary	Fio+Predm+STR(Gruppa)
T_PREP	i_fio	Primary	Fio
T_PREP	i_dolgn	Candidate	Dolgn
T_PREP	i_stag	Candidate	Stag
T_STAG	i_stag	Primary	Stag
T_DOLGN	i_dolgn	Primary	dolgn

После создания всех необходимых индексов в окне Конструктора БД (рис. 12.4) будут представлены все таблицы в виде списка из полей и индексов, разделенных ключевым словом **Indexes** (индексы). Если в таблице список полей и индексов виден не полностью, его можно прокрутить вниз или вверх. На рис. 12.5 показаны установленные межтабличные связи.

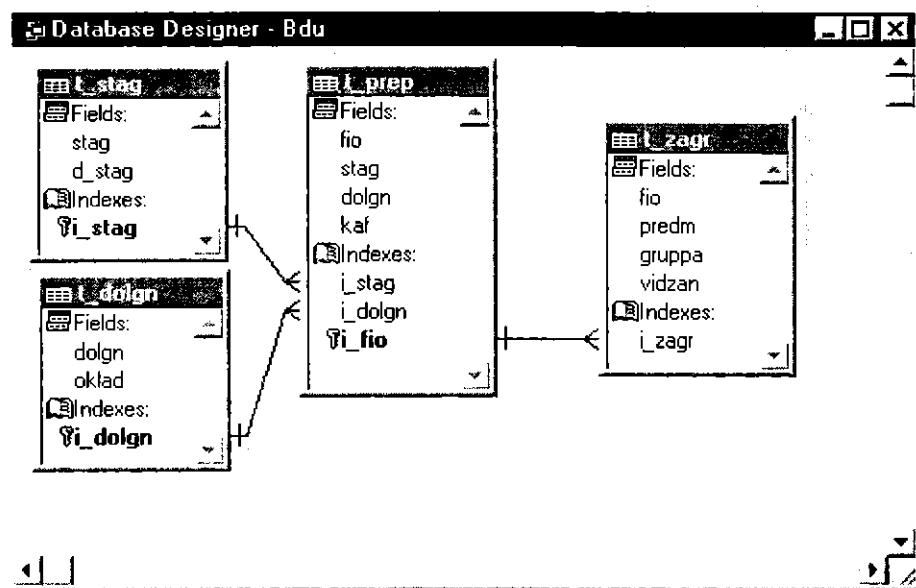


Рис. 12.5. Межтабличные связи БД Bdu

Для установления постоянной связи между родительской и дочерней таблицей необходимо:

- установить указатель мыши на первичный ключ родительской таблицы;
- нажав левую кнопку мыши, переместить указатель на индекс дочерней таблицы, по которому устанавливается связь;
- отпустить кнопку.

На экране появится диалоговое окно **Edit Relationship** (правка связей), содержащее имена связываемых таблиц с раскрывающимися списками индексов. В списках уже выбраны индексы, по которым должна осуществляться связь. Здесь же указывается тип связи между таблицами «один к одному» или «один ко многим». Для сохранения связи нажмем **OK**, а при отказе — **Cancel** (отмена).

Для удаления межтабличной связи требуется выполнить следующее:

- навести указатель мыши на линию связи таблиц;
- нажать правую кнопку мыши, что приведет к утолщению линии и появлению всплывающего меню;
- выполнить команду **Remove Relationship** (удалить связь) всплывающего меню.

В результате выделенная связь будет удалена. Напомним, что установленные постоянные связи между таблицами используются прежде всего для обеспечения ссылочной целостности (непротиворечивости) БД при обновлении ее содержимого. Для снижения возможных потерь данных рекомендуется сначала устанавливать правила ссылочной целостности, а затем вводить данные в таблицы.

12.9. Обеспечение ссылочной целостности

Обеспечение ссылочной целостности означает определение допустимых операций над связанными между собой таблицами. Главное требование ссылочной целостности заключается в том, чтобы записи дочерних таблиц имели ссылки на записи в родительской таблице (дети должны иметь родителей). В родительской таблице могут быть записи, не имеющие связанных с ними записей в дочерних таблицах (может быть семья без детей). В Visual FoxPro поддержка ссылочной целостности выполняется с помощью одноименного Построителя. Чтобы открыть Построитель ссылочной целостности, достаточно выполнить следующее:

- открыть БД с таблицами, для которых необходимо установить ссылочную целостность;
- в контекстном меню Конструктора БД задать команду **Referential Integrity** (ссылочная целостность), что и приведет к открытию окна Построителя ссылочной целостности данных (рис. 12.6).

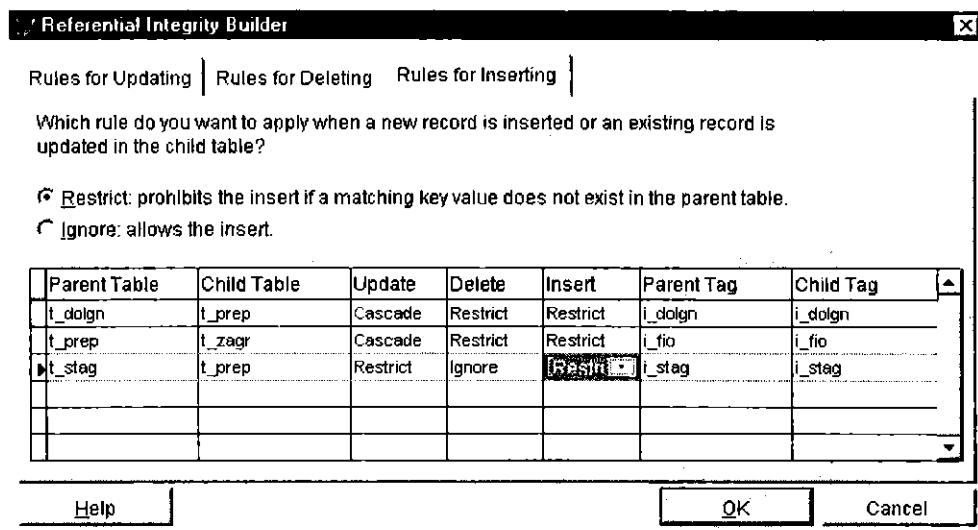


Рис. 12.6. Диалоговое окно Referential Integrity Builder

В окне Построителя перечислены все постоянные связи между таблицами БД Bdu. Информация по каждой связи размещена в отдельной строке. Первые два столбца содержат имена родительских и дочерних таблиц. Следующие три столбца содержат правила поддержания ссылочной целостности в случае редактирования, удаления и добавления записей соответственно. Изначально все правила содержат Ignore, но они определяются для каждой связи и каждой из трех перечисленных операций. Последние два столбца содержат индексы (теги) родительской и дочерней таблиц, по которым осуществляется связь.

В диалоговом окне Построителя ссылочной целостности корректируем могут быть подвергнуты только столбцы с правилами. Наша задача — для каждой связи (пары родительской и дочерней таблицы) определить правила поддержания ссылочной целостности при выполнении каждой из трех возможных операций с данными этих таблиц (редактирования, добавления и удаления). Этот выбор можно осуществить двумя способами:

- при выборе поля одного из столбцов с правилами **Update** (обновить), **Delete** (удалить) и **Insert** (вставить) появляется кнопка, нажатие которой приводит к появлению списка возможных правил;
- при выборе одной из вкладок **Rules for Updating** (правила обновления), **Rules for Deleting** (правила удаления), **Rules for Inserting** (правила вставки) в верхней части таблицы появляются переключатели с комментариями для каждого из правил.

Например, при выборе вкладки **Rules for Updating** (правила обновления) выбираются правила, связанные с изменениями значения первичного ключа

или ключа-кандидата в родительской таблице. При этом возможен выбор одного из трех правил, приведенных в табл. 12.7.

Таблица 12.7
Правила изменения значений первичного ключа

Наименование	Описание
Cascade	При изменении значений полей первичного ключа или ключа-кандидата в родительской таблице автоматически осуществляется изменение всех соответствующих значений в дочерней таблице — каскадное изменение.
Restrict	Запрещается изменение первичного ключа или ключа-кандидата в родительской таблице, если в дочерней таблице имеется хотя бы одна запись, внешний ключ которой содержит изменяемое значение.
Ignore	Допускается произвольное изменение значений полей первичного ключа или ключа-кандидата родительской таблицы. Целостность данных не поддерживается.

Выбор любого из правил осуществляется установкой соответствующего переключателя. Выбранное правило отображается в соответствующей строке столбца **Update** (обновить).

При удалении записей из связанных таблиц (вкладка **Rules for Deleting** — правила удаления) требуется использование также трех правил (табл. 12.8).

Таблица 12.8
Правила удаления записей из связанных таблиц

Наименование	Описание
Cascade	Удаление записей в родительской таблице автоматически приводит к каскадному удалению всех записей дочерней таблицы, имеющих аналогичные значения соответствующего внешнего ключа.
Restrict	Запрещается удаление записи в родительской таблице, если в дочерней таблице имеется хотя бы одна запись, внешний ключ которой содержит значение, совпадающее со значением первичного ключа или ключа-кандидата в удаляемой записи. При попытке удаления записи выдается сообщение об ошибке, которую можно обработать программно.
Ignore	Допускается неограниченное удаление записей родительской таблицы. Целостность данных не поддерживается.

Правила для добавления записей применимы со стороны дочерней таблицы. Их всего два (табл. 12.9).

Таблица 12.9
Правила добавления записей

Наименование	Описание
Restrict	Запрещается добавление записи к дочерней таблице, если в родительской таблице отсутствует запись, значение первичного ключа или ключа-кандидата которой не совпадает со значением внешнего ключа добавляемой записи.
Ignore	Допускается неограниченное добавление записей в дочернюю таблицу. Целостность данных не поддерживается.

По окончании выбора правил выйдем из Построителя, нажав ОК. При выходе появляется запрос подтверждения сделанных изменений. В случае подтверждения в БД создается ряд хранимых процедур и триггеров. Если такие процедуры и триггеры уже существуют, то перед их перезаписью создается резервная копия. Теперь все наши требования по обеспечению ссылочной целостности данных сохраняются в словаре БД и реализуются при работе с ней.

При формировании правил ссылочной целостности (рис. 12.6) мы рассуждали следующим образом:

1. Изменение значений ключевых полей родительских таблиц. Таблица T_Pger является дочерней по отношению к таблицам T_Dolgn и T_Stag, в свою очередь, для нее дочерней является таблица T_Zagr. Не будем исключать возможность изменения названия должности или фамилии преподавателя. Соответствующие поля являются ключевыми. При необходимости такие изменения необходимо произвести в родительских таблицах соответственно: должность в – T_Dolgn, а фамилию в – T_Pger. В этом случае они синхронно (каскадно) изменятся в дочерних таблицах. Запретим изменения значений ключевого поля stag (стаж) в родительской таблице T_Stag.
2. Удаление записей в родительских таблицах. Запретим удаление записей в родительских таблицах, имеющих в дочерних таблицах записи с совпадающими значениями внешних ключевых полей.
3. Изменение (добавление) записей в дочерней таблице. Запретим ввод записей в дочерние таблицы, не соответствующие одной из записей в родительской таблице.

Проверим средства обеспечения ссылочной целостности в действии.

Откроем БД Bdu с помощью команды File | Open (файл | открыть) системного меню Visual FoxPro (если она закрыта). В появившемся окне Конструктора БД откроем для просмотра таблицу T_rger. Для этого установим курсор на эту таблицу и, вызвав всплывающее меню, зададим команду Browse (просмотр).

Выполним изменение, например, название должности (преп на ст. преп.) сначала в дочерней, а затем в родительской таблице. Изменения будут видны только после обновления изображения таблицы (закрытия и повторного открытия таблицы).

Замечание.

При необходимости произвести добавление или изменение защищенных данных (режим Restrict) нужно снять защиту и после выполнения требуемых действий повторно установить защиту.

12.10. Создание запросов

После создания таблиц БД и ввода в них данных требуется организовать доступ к хранящейся в БД информации для просмотра и обработки. Одним из основных способов решения названной задачи является создание запросов.

Средства формирования запросов

Выборка информации из БД может осуществляться: с помощью команды SELECT SQL языка Visual FoxPro, которая является аналогом соответствующей команды языка SQL; с помощью Мастера запросов и с помощью Конструктора запроса.

Команда SELECT имеет множество возможностей (опций). Ее упрощенное представление имеет следующий вид:

```
SELECT СписокВыбираемыхПолей  
      FROM СписокТаблиц-источника данных | INTO ИмяТаблицы получателя данных |  
           [WHERE УсловиеВыборки]  
           [GROUP BY УсловиеГруппировки]  
           [ORDER BY УсловиеУпорядочивания выводимых данных]  
           [TO FILE ИмяФайла | TO PRINTER – направление вывода данных]  
Квадратные скобки указывают на необязательность опции.
```

Конструктора запроса позволяет:

- выбирать данные из одной или нескольких таблиц, используя сложные критерии;

- устанавливать временные связи между таблицами;
- выбирать поля и записи таблиц с требуемыми данными;
- выполнять вычисления с использованием выбранных данных.

Работа с Конструктором запроса сводится к заполнению форм запроса. Результатом запроса всегда является таблица, которая может быть сохранена в массиве или в созданной новой таблице, отображена на экране или оформлена в виде отчета. Данные, содержащиеся в результирующей таблице, могут быть представлены в виде графика или гистограммы. Рассмотрим формирование запросов с помощью Конструктора запроса.

Для вызова Конструктора запроса после открытия БД можно воспользоваться командой **File | New** (файл | Создать) системного меню Visual FoxPro. При выполнении этой команды открывается диалоговое окно **New** (рис. 12.2), в котором нужно выбрать переключатель **Query** (Запрос) и нажать кнопку **New File** (Новый файл). В открывшемся диалоговом окне **Add Table or View** (Добавить таблицу или просмотр) следует выбрать одну или несколько таблиц и нажать **OK**. В результате откроется окно Конструктора запроса (рис. 12.7), содержащее выбранные таблицы, а в системном меню Visual FoxPro появится пункт **Query** (Запрос).

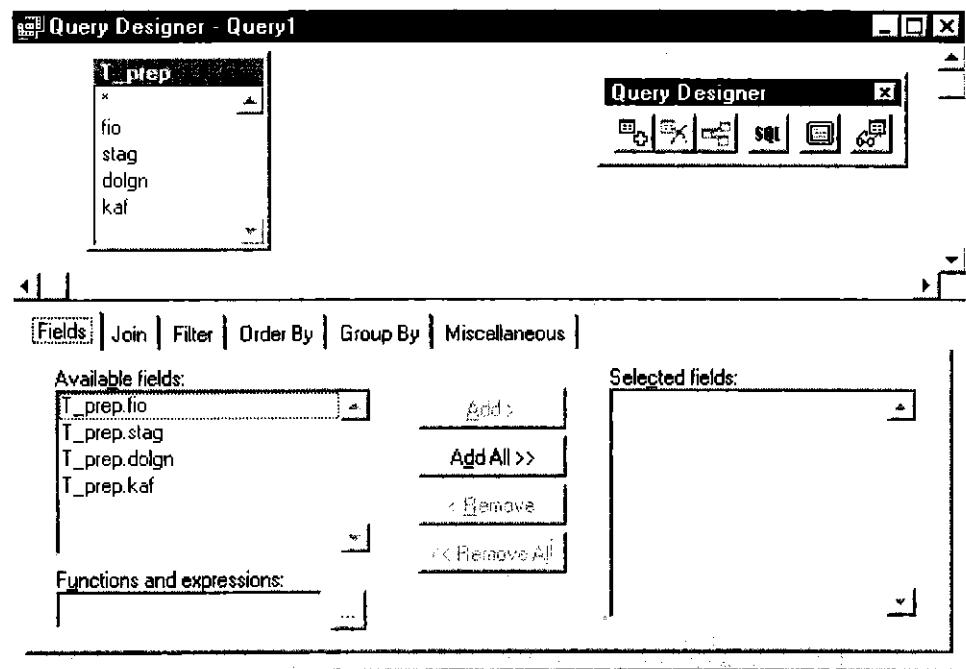


Рис. 12.7. Окно Конструктора запроса с таблицей T_PREP

После того как определена таблица (одна или несколько) для формирования запроса в диалоговом окне Конструктора необходимо:

- выбрать поля, содержащие искомые данные;
- задать критерии выборки, упорядочения и группировки данных;
- указать, куда выводится результат запроса.

Для решения перечисленных задач Конструктор содержит панель для отображения таблиц, используемых в запросе (в рассматриваемом примере на ней находится одна таблица *T_PREP*), и вкладки (табл. 12.10).

Таблица 12.10
Назначение вкладок диалогового окна Конструктора запросов

Вкладка	Назначение
Fields	Выборка полей в результирующую таблицу
Join	Задание условий объединения таблиц
Filter	Задание фильтра (условий отбора записей)
Order By	Задание критерия упорядочивания
Group By	Задание условий группировки данных
Miscellaneous	Дополнительные установки (признак выборки повторяющихся значений, количество выбираемых данных)

Поле *Functions and expressions* (функции и выражения) используется для включения в запрос функции или выражения.

По окончании формирования запроса в диалоговом окне Конструктора результат выполнения запроса можно оценить выполнив команду *Query | Run Query* (Запрос | выполнить запрос) системного меню.

Для повторного выполнения запроса, сформированного в окне Конструктора, его необходимо сохранить. Для этого достаточно выполнить команду *File | Save* (Файл | сохранить) и в открывшемся окне указать полное имя файла, в котором будет сохранено окно с описанием запроса. В результате будет создан файл, хранящий запрос, с расширением *QPR*. Теперь для повторного выполнения запроса достаточно открыть этот файл командой *File | Open* (Файл | открыть) и запустить на выполнение командой *Query | Run Query* (Запрос | выполнить запрос).

Выбор полей результирующей таблицы

Выбор полей результирующей таблицы выполняется с помощью вкладки *Fields* (Поля) диалогового окна Конструктора запросов. Рассмотрим эту процедуру на примере организации запроса 1.

Запрос 1. Составить список всех преподавателей, сведения о которых содержатся в таблице T_PREP.DBF, с указанием их стажа и должности.

Вся необходимая информация содержится в одной таблице. Выводимые записи должны быть упорядочены по полю Fio.

Поля результирующей таблицы формируются из полей исходной таблицы и вычисляемых полей. Для выбора полей исходной таблицы необходимо отобразить их в списке **Selected Output** (выбранные поля) вкладки выбора полей **Fields** (поля) (рис. 12.7). Один из способов формирования полей в результирующей таблице состоит в следующем. В окне Конструктора запросов выберем вкладку **Fields** (поля) и откроем при этом два списка: **Available Fields** (доступные поля) и **Selected Output** (выбранные поля) (рис. 12.7). Наша задача перенести в список **Selected Output** поля, используемые в запросе. При выделении в списке **Available Fields** (доступные поля) с помощью мыши нужного поля активизируется кнопка **Add** (добавить), при нажатии которой осуществляется перенос этого поля. Для переноса всех полей в список выбранных достаточно нажать кнопку **Add All** (добавить все). Если часть полей оказалась лишней, их можно удалить из списка **Selected Output** (выбранные поля) с помощью кнопки **Remove** (удалить).

Место, которое поле занимает в списке **Selected Output** (выбранные поля), соответствует и его месту в результирующей таблице. Для изменения расположения некоторого поля достаточно выбрать мышью маркер перемещения (слева от поля) и переместить его в нужное место.

Результаты выполнения запроса представлены на рис. 12.8.

	Fio	Stag	Dolgn	Kaf
	Иванов И.И.	5	преп	25
	Петров П.П.	7	преп	25
	Сидоров С.С.	10	доцент	25
	Егоров Е.Е.	12	профессор	25
	Баглюк С.И.	12	ст. преп	24

Рис. 12.8. Результат выборки полей таблицы T_prep

Рассмотрим процедуру *упорядочения данных* в таблице на примере следующего запроса.

Запрос 2. Составить список всех преподавателей из таблицы T_PREP.DBF с указанием их стажа и должности. Фамилии в списке упорядочить по алфавиту.

Для упорядочения данных используется вкладка **Order By** (упорядочение). В ней содержится два списка **Selected Fields** (выбранные поля) и

Ordering Criteria (критерий упорядочения). Для задания критерия упорядочивания необходимо перенести в Ordering Criteria (критерий упорядочения) поля, которые будут определять порядок расположения выводимых в запросе данных. Процедура переноса аналогична рассмотренной при определении полей результирующей таблицы. Для реализации нашего первого запроса во вкладке Order By (упорядочение) выделим курсором поле T_prep.fio (поле fio таблицы T_prep) и, нажав кнопку Add (добавить), перенесем его в список Ordering Criteria (критерий упорядочения).

С помощью переключателя Order Options (параметры упорядочения) для каждого выбранного поля можно установить критерий упорядочивания по возрастанию (Ascending) или по убыванию (Descending). Выберем критерий Ascending (возрастающий). Результат выполнения подготовленного описанным образом запроса будет аналогичен представленному на рисунке 12.8, но фамилии преподавателей будут следовать в алфавитном порядке.

Фильтрация записей

До сих пор в качестве результата запроса мы получали все записи исходной таблицы. Чаще возникает необходимость в просмотре части записей таблицы, удовлетворяющих определенным условиям. Процедуру отбора записей, отвечающих определенному условию, называют процедурой фильтрации записей. Для ее выполнения используется вкладка Filter (фильтр) Конструктора запроса. Рассмотрим эту процедуру на примере следующего запроса.

Запрос 3. Выполнить запрос, аналогичный запросу 2, но сформировать список сведений только о преподавателях 25 кафедры.

Поля, входящие в условия отбора записей, не обязательно должны быть включены в запрос.

Для задания условия фильтрации записей выберем вкладку Filter (фильтр). Из раскрывающегося списка полей исходной таблицы в столбце Field Name (имя поля) выберем поле T_prep.kaf для отбора. В списке Criteria (критерий) выберем оператор точного сравнения (= =). Далее зададим значение поля kaf, по которому осуществляется сравнение. Для этого в поле Example (пример) введем 25. Рассматриваемый в примере запрос готов к выполнению.

В раскрывающемся списке Criteria (критерий) содержатся следующие операторы сравнения:

- = — равенство,
- Like — вхождение,
- = = — фактическое равенство,
- > — больше чем,
- < — меньше чем,

\geq — не меньше чем,
 \leq — не больше чем,
Is NULL — совпадение с NULL,
Between — в диапазоне значений,
In — среди заданных значений.

Оператор = (Equal) позволяет осуществлять поиск при условии знания части первых символов в значениях поля, по которому осуществляется поиск. Эта часть символов вводится в поле столбца Example (пример) и сравнивается со значением поля, указанного в столбце Field Name (имя поля).

Оператор Like (подобие) выполняет посимвольное сравнение строки, стоящей слева, со строкой, находящейся справа, пока она не закончится.

Операторы $>$, \geq , $<$, \leq и Between (между) можно использовать с текстовыми, цифровыми полями и полями дат. Например, для выборки преподавателей со стажем более 10 лет, достаточно в качестве условий отбора выбрать поле Stag, в списке вариантов сравнения выбрать оператор \geq , а в поле Example (пример) ввести 10.

При использовании оператора Between (между) начальное и конечное значение вводится через запятую. При задании диапазона для значений текстовых полей сравниваются коды символьных величин. Так, условие выбора «А, Д», помещенное в поле Example (пример) применительно к полю Fio позволит выбрать из таблицы Т_Prep всех тех преподавателей, фамилии которых начинаются с букв в данном диапазоне.

При необходимости выбора сведений о конкретных преподавателях следует использовать оператор In. В этом случае в поле столбца Example (пример) через запятую перечисляются фамилии преподавателей.

Для реализации запроса с отрицанием условий сортировки, выбранных в списке Criteria, используется размещенный перед списком флажок Not.

Возможно формирование критерия отбора записей, состоящего из нескольких условий, соединяемыми операторами AND (И) или OR (ИЛИ) из списка Logical.

Кнопки Insert (вставить) и Remove (удалить) соответственно добавляют и удаляют условия из списка условий отбора записей. Чтобы вставить новое условие между существующими условиями, необходимо выделить нижнее из них и нажать кнопку Insert (вставить). В этом случае будет добавлена пустая строка перед помеченной.

В качестве критерия отбора может выступать сложное выражение, включающее не только поля таблиц, но также переменные и выражения. Для формирования подобных выражений может использоваться построитель выражений Expression Builder, который вызывается при выборе элемента <Expression...> (<выражение...>) в списке Field Name (имя поля).

Организация многотабличных запросов

Для формирования многотабличного запроса необходимо разместить в окне Конструктора запросов все участвующие в запросе таблицы и определить условия их связи. Вместе с таблицами в окне Конструктора запросов отображаются и их постоянные связи. Если необходимые постоянные связи отсутствуют, то с помощью вкладки **Join** (связь) организуются временные межтабличные связи.

Рассмотрим создание многотабличного запроса на следующем примере.

Запрос 4. Необходимо получить на экране информацию о преподавателях кафедры 25 их должностях и окладах, преподавательском стаже и надбавке за стаж.

Требуемая информация содержится в следующих трех таблицах: T_prep, T_dolgn, T_stag. В окне Конструктора запросов уже имеется таблица T_prep. Остальные можно добавить, например, с помощью команды **Query | Add Table** (запрос | добавить таблицу). При этом в появившемся диалоговом окне **Add Table or View** (добавить таблицу или запрос) выбирается требуемая таблица. В результате образ таблицы появляется в окне Конструктора запросов.

Если при создании БД между выбранными таблицами были установлены постоянные связи, то они будут также представлены, во-первых, в виде линий, соединяющих соответствующие поля в образах этих таблиц (верхняя часть рис. 12.9), а во-вторых — во вкладке **Join** добавляются соответствующие строки с условиями объединения таблиц.

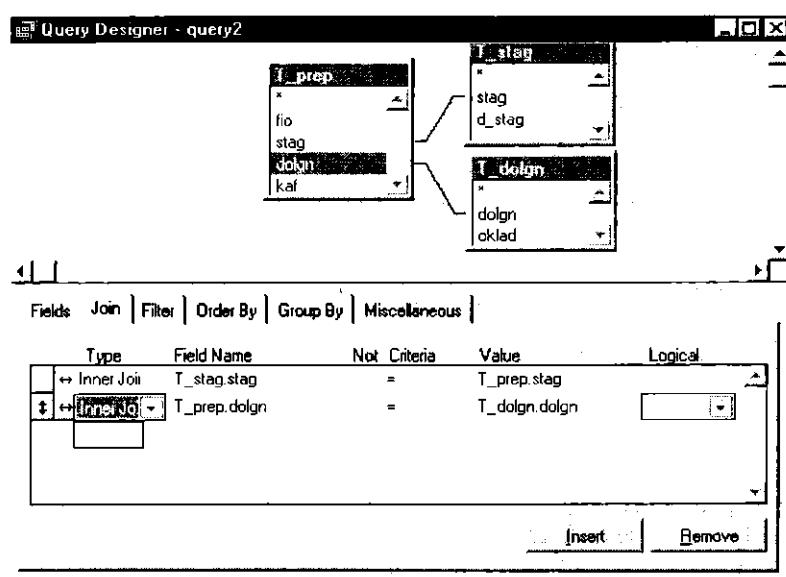


Рис. 12.9. Окно многотабличного запроса с выбранной вкладкой **Join**

Если ранее постоянных связей между выбранными таблицами установлено не было, то открывается диалоговое окно **Join Condition** (условие связи), служащее для установления или модификации связи (рис. 12.10). В связи с тем, что в рассматриваемом примере имеются постоянные связи между выбранными таблицами, то окно **Join Condition** (условие связи) было вызвано двойным щелчком мыши на выбранной линии, связывающей таблицы по полю **dolgn**. Этот прием используется также при модификации типа связи.

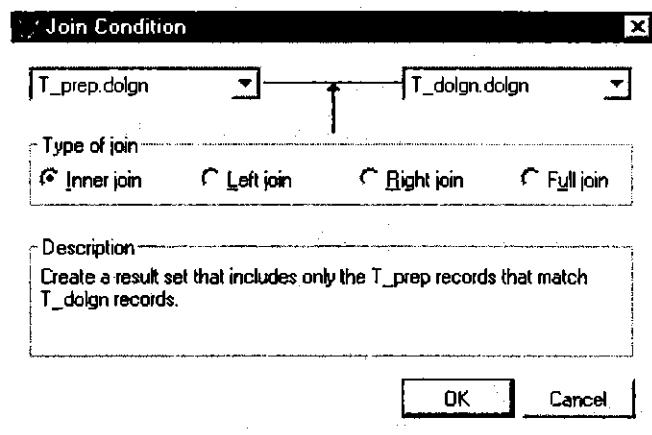


Рис. 12.10. Окно **Join Condition**

Тип связи модифицируется или задается с помощью переключателя **Type of join** (тип связи) рассматриваемого окна. В левой и правой частях окна содержатся раскрывающиеся списки полей двух таблиц, с помощью которых выбираются поля для связи таблиц. Будем использовать предлагаемую по умолчанию опцию **Inner join**. В этом случае создается объединение, в которое выбираются записи, которые содержат совпадающие значения в полях связи.

Замечание.

Для установления временных связей возможно использование любых полей таблиц без ограничений, имеющих место при установке постоянных межтаблицевых связей.

Формирование вычисляемого поля в запросе

В качестве полей результирующей таблицы могут использоваться вычисляемые поля. Вычисляемое поле представляет собой выражение, включающее одно или несколько полей исходной таблицы, константы и функции, соединенные операторами. Для включения в запрос вычисляемого поля необходимо выбрать вкладку **Fields** Конструктора запроса и в поле **Function**

and Expressions (функция и выражения) (рис. 12.1) ввести необходимое выражение. Далее после нажатия кнопки Add (добавить) данное выражение будет занесено в список полей запроса.

Если нажать кнопку справа от поля Function and Expressions (функция и выражения), то открывается диалоговое окно Построителя выражения — Expression Builder (рис. 12.11), упрощающего формирование выражения. Выбирая в поле From table (из таблицы) Построителя таблицу и в поле Fields (поля) — название поля, формируем требуемое выражение, которое при этом выводится в поле Expression (выражение). По нажатию кнопки OK вычисляемое поле переносится в список полей запроса.

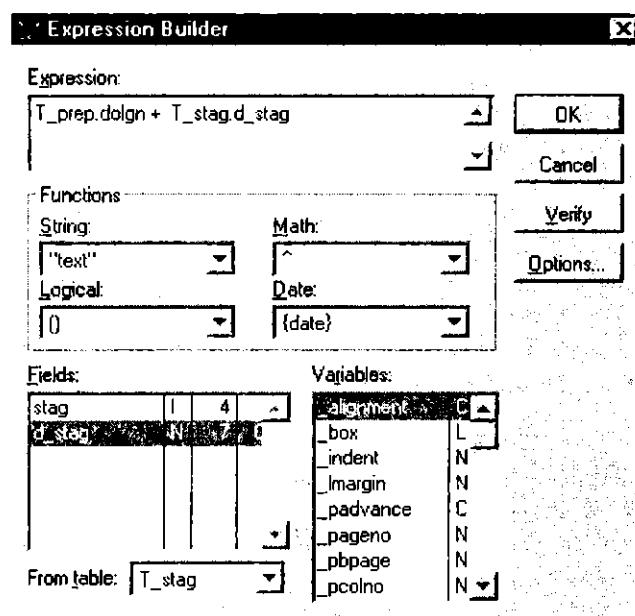


Рис. 12.11. Диалоговое окно Expression Builder

Результат выполнения многотабличного запроса с вычисляемым полем (Exp_7) представлен на рис. 12.12.

	Fio	Kaf	Dolgn	Stag	D_stag	Oklad	Exp_7
1	Иванов И.И.	25	преп.		5	50	1000
2	Петров П.П.	25	преп.		7	50	1050
3	Сидоров С.С.	25	доцент		10	100	2000
4	Егоров Е.Е.	25	профессор		12	150	2650

Рис. 12.12. Результат выполнения многотабличного запроса

Из рисунка 12.12 видно, что вычисляемому полю автоматически был присвоен заголовок (Exp_7), который не очень содержателен. Есть возможность изменять заголовки полей в запросе. Для этого в поле **Function and Expressions** (функция и выражения) Конструктора запроса к названию формируемого поля добавляется ключевое слово AS и новый заголовок.

Сохраним результаты запроса в файле Query4.QPR.

Примеры сложного упорядочения данных

В качестве примера сложного упорядочения данных рассмотрим создание следующего запроса.

Запрос 5. Организовать выборку сведений о преподавателях со стажем работы от 5 до 10 лет. Сведения упорядочить по виду занятия и фамилии преподавателя.

Выборка информации должна осуществляться из всех четырех таблиц.

Нам необходимо организовать сложный запрос, позволяющий упорядочить выбранные сведения сначала по видам занятий, а затем по фамилии преподавателя. Для этого во вкладке **Order By** (упорядочение) в список **Ordering Criteria** (критерий упорядочения) необходимо занести сначала поле VidZan, а затем Fio. Для задания выборки по заданному диапазону преподавательского стажа укажем во вкладке **Filter** его нижнюю (5 лет) и верхнюю (10 лет) границы.

Подготовим запрос следующим образом.

1. Создадим новый запрос с помощью команды **File | New** (файл | создать).
2. Добавим в окно Конструктора запроса все таблицы БД.
3. На вкладке **Fields** (поля) занесем в список **Selected fields** (выбранные поля) выбираемые поля: T_Prep.Fio, T_Prep.Stag, T_Zagr.VidZan, T_Zagr.Gruppa.
4. На вкладке **Filter** (фильтр) зададим условия отбора записей. Для этого в списке **Field Name** (имя поля) укажем поле, по которому будет осуществляться выборка, — T_Prep.Stag. В списке **Criteria** (критерий) выберем оператор сравнения **Between** (между), задающий условия выборки. В поле **Столбца Example** (пример) введем через запятую начальную и конечную границы выборки 5, 10.
5. Зададим условия упорядочения записей. Для этого выберем вкладку **Order By** (упорядочение) и в список **Ordering Criteria** (критерий упорядочения) внесем последовательно поля T_Zagr.VidZan и T_Prep.Fio (рис. 12.13).

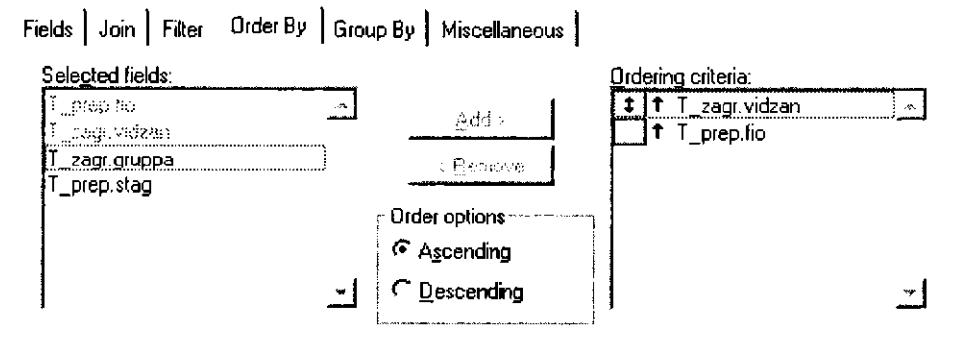


Рис. 12.13. Выборка со сложным упорядочиванием данных

Для выполнения запроса нажмем кнопку Run (выполнить).

Представление результатов запросов

В рассмотренных примерах результат запроса представлялся в табличном виде на экране. Между тем, результат запроса можно запомнить во вновь созданной таблице или файле, преобразовать в отчет или направить на принтер. Чтобы реализовать одну из перечисленных возможностей, достаточно выполнить команду **Query | Query Destination** (запрос | назначение запроса) и в появившемся диалоговом окне **Query Destination** (назначение запроса) (рис. 12.14) нажать нужную кнопку (табл. 12.11).

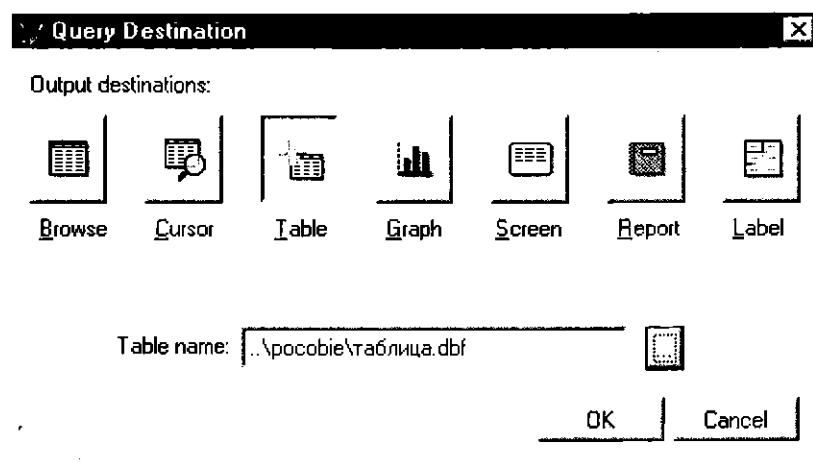


Рис. 12.14. Диалоговое окно Query Destination

Таблица 12.11

Кнопки выбора варианта представления результата запроса

Кнопка	Назначение
BROWSE	Просмотр результатов в окне Browse
CURSOR	Временное хранение результата запроса для просмотра
TABLE	Сохранение в виде таблицы
GRAPH	Создание диаграммы с помощью MS Graph
SCREEN	Вывод результатов выборки в активном окне
REPORT	Представление в виде отчета
LABEL	Представление в виде этикетки

Результаты всех реализованных ранее запросов представлялись в окне **Browse** (просмотр) – по умолчанию, нажата кнопка **Browse** (просмотр). При выборе этого режима создается временная таблица, которая и выводится в окне **Browse** (просмотр). Таблица удаляется из памяти вместе с закрытием окна, поэтому данный режим используется тогда, когда результаты запроса необходимо только просмотреть.

Рассмотрим оформление в виде отчета результатов многотабличного запроса 4, используя сохраненный ранее файл запроса **Query4.QPR**. Для упрощения решения задачи создания отчета на основе многотабличного запроса выполним следующее.

1. Представим результаты запроса в виде одной таблицы, используя опцию **Table** (таблица) окна **Query Destination** (назначение запроса).
2. Создадим отчет с помощью Мастера однотабличных отчетов.

Сохранение многотабличного запроса и оформление отчета

При выборе параметра **Table** (таблица) в окне **Query Destination** (назначение запроса) (рис. 12.14) мы заказываем форму представления результата запроса в виде таблицы и указываем имя файла, в котором она в дальнейшем будет сохранена. Задание имени файла таблицы в окне **Query Destination** (назначение запроса) выполняется путем ввода имени файла в поле **Table name** (имя таблицы) или с помощью диалогового окна **Open** (открыть), вызываемого нажатием кнопки справа.

Замечание.

Все опции направления вывода результатов, выбираемые в диалоговом окне **Query Destination** (назначение запроса), дают ожидаемый результат только после нажатия в нем кнопки **OK** и выполнения запроса, например с помощью команды **Query | Run Query** (запрос | выполнить запрос).

Выполним запрос, результат которого будет сохранен в файле с указанным ранее именем. Созданная таблица сохраняется и при выходе из Visual FoxPro. В этой таблице можно производить удаление, добавление и редактирование записей.

После создания таблицы можно приступить к формированию отчета на ее основе. Для этого вызовем диалоговое окно **Query Destination** (назначение запроса), выполнив одноименную команду меню **Query** (запрос). В появившемся окне нажмем кнопку **Report** (отчет). В результате получим соответствующий вариант диалогового окна **Query Destination** (назначение запроса) (рис. 12.14). Чтобы вызвать Мастер отчетов, нажмем кнопку с «волшебной палочкой». На экране появится диалоговое окно **Report Wizard** (Мастер отчетов).

Замечание.

Для обеспечения возможности вызова Мастера отчетов необходимо предусмотреть указание его местоположения (папки и имени приложения) в диалоговом окне **Options** (Параметры) на вкладке **File Locations** (Расположение файлов) в поле **Wizards** (Мастера). Вызов окна выполняется с помощью команды **Tools | Options** (сервис | параметры).

Работа с Мастером отчетов заключается в выполнении пяти следующих шагов.

1. Выбор таблицы (одной), на основе которой будет формироваться отчет, и выбор полей таблицы, сведения из которых будут включены в отчет. Для решения этих задач в списке **Databases and tables** (базы данных и таблицы) выбирается одно из двух значений: **DBU** (таблица будет выбираться из БД) или **Free Tables** (таблица является свободной). В нашем случае необходим второй вариант, так как выбирается таблица с результатами запроса. Далее формируем список **Selected fields** (выбранные поля). Переносим в него поля таблицы из списка **Available fields** (доступные поля) в том порядке, в каком хотим их видеть в отчете.
2. Выбор стиля оформления отчета.
3. Выбор ориентации листа (книжной или альбомной) и способа расположения сведений из полей таблицы (в строке или в колонке).
4. Задание способа сортировки сведений в отчете: имя поля, по которому осуществляется сортировка; способ сортировки — по возрастанию или по убыванию.
5. Выполнение следующих действий: формирование заголовка отчета; указание способа его дальнейшего использования (мы выбрали запись отчета в файл); при необходимости предварительный просмотр отчета (опция **Preview**); завершение подготовки отчета нажатием кнопки **Finish** (финиш).

Результат предварительного просмотра отчета представлен на рис. 12.15.

Фамилия	Должность	Оклад	Д За Стаж	Сумма
Иванов И.И.	преп	1,000	50	1,050
Петров П.П.	преп	1,000	50	1,050
Сидоров С.С.	доцент	2,000	100	2,100
Егоров Е.Е.	профессор	2,500	150	2,650

Рис. 12.15. Предварительный просмотр отчета

После нажатия кнопки **Finish** на экране вновь появится диалоговое окно **Query Destination** с предложением подтвердить необходимость сохранения созданного отчета в файле.

Контрольные вопросы и задания

1. Дайте общую характеристику СУБД Visual FoxPro.
2. Назовите требования, предъявляемые Visual FoxPro 8.0 к аппаратным ресурсам.
3. Перечислите состав панелей инструментов в главном окне СУБД.
4. Укажите назначение элементов проекта Visual FoxPro.
5. Какие средства автоматизации разработки проекта имеются в Visual FoxPro, в чем различие их возможностей?
6. Перечислите типы данных, используемые в Visual FoxPro.
7. Перечислите типы индексов.
8. Где и зачем устанавливаются постоянные связи между таблицами, и где они хранятся?
9. Перечислите правила обеспечения ссылочной целности, используемые в Visual FoxPro.

Литература

1. Агальцов В. П. Базы данных. – М.: Мир, 2002.
2. Горев А. Visual FoxPro 5.0. Книга для программистов. – М: Журнал "FoxTalk" ТОО «Эдель», 1997. – 552 с.
3. Михаель Д. Антонович и др. Visual FoxPro 3 для Windows. -- М.: БИНОМ, 1996. – 668 с.
4. Омельченко Л. Visual FoxPro 8. – СПб.: БХВ-Петербург, 2003. – 688 с.
5. Омельченко Л., Шевякова Д. Visual FoxPro 9.0. – СПб.: БХВ-Петербург, 2005.

13. Microsoft SQL Server 2000

Microsoft SQL Server представляет собой СУБД, обеспечивающую создание информационных систем с архитектурой «клиент-сервер», в которой он играет роль сервера баз данных. SQL Server удовлетворяет требованиям, предъявляемым к системам распределенной обработки информации. Эта СУБД поддерживает: тиражирование данных, параллельную обработку, создание и обработку больших баз данных на недорогих аппаратных платформах, отличается простотой управления и использования, а также обеспечивает тесную интеграцию баз данных SQL Server 2000 в Web.

Наиболее важным нововведением можно считать интеграцию баз данных в Web. Используя язык XML, пользователи могут легко опубликовать информацию из баз данных в Интернете, обеспечив при этом возможность доступа к данным с помощью обычного обозревателя (браузера). Кроме того, изменения в архитектуре сервера позволили полностью интегрировать SQL Server 2000 со службами Active Directory операционной системы Windows 2000. Дадим краткую характеристику рассматриваемому серверу и введем основные понятия.

13.1. Характеристика SQL Server

Основные новшества сервера

Нами рассматривается SQL Server версии 2000. Несмотря на то, что в сравнении с предыдущей версией ядро системы практически сохранилось, в ней имеется достаточно большое число существенных новшеств. Рассмотрим кратко эти изменения.

Поддержка множества инсталляций. Теперь на одном компьютере можно устанавливать несколько копий (инсталляций) серверов БД SQL Server 2000, каждая из которых является независимой от других с точки зрения управления. Инсталляции имеют свои имена (*инсталляция по умолчанию* может не иметь имени), могут запускаться и останавливаться отдельно от других. Независимость инсталляций достигается тем, что каждая из них имеет свой набор служб операционной системы (MSSQLServer и SQLServerAgent), а также собственный набор пользовательских и системных БД.

Сопоставления. Поведение сервера SQL Server 7.0 в операциях сравнения и сортировки задавалось параметрами кодовой страницы, порядка сортировки и сопоставления Unicode. Причем, делалось это только на уровне всего

сервера, то есть действовали они одинаково на все столбцы всех таблиц всех БД сервера.

В новой версии продукта пользователь выбирает сопоставление (collation), включающее в себя необходимые параметры и может это делать на различных уровнях: для сервера, базы данных, таблицы и даже отдельного столбца. Сопоставление, конфигурируемое на уровне сервера, является сопоставлением по умолчанию и действует в случаях, когда на определено другое сопоставление.

Определяемые пользователем функции. Ранее в MS SQL Server имелся фиксированный набор встроенных функций, используемых в выражениях. В сложных алгоритмах пользователю приходилось применять курсоры и другие механизмы для вызова хранимых процедур (где производилась основная обработка информации). Теперь имеется возможность создания своих функций, вызываемых из тела запроса.

Расширение возможностей триггеров. Триггеры, создаваемые ранее только для таблиц, теперь можно использовать и для представлений, что существенно расширяет функциональность сервера. В предыдущих версиях существовал только один тип триггера (тип AFTER), который запускался после соответствующей операции. При необходимости отмены произведенных действий приходилось использовать откат транзакций. Триггеры нового введенного типа (тип INSTEAD OF) работают вместо команды пользователя, приведшего к запуску триггера.

Улучшения в индексах. Наряду с обычным индексированием таблиц допускается индексирование представлений и индексирование вычисляемых столбцов (computed columns). Это позволяет повысить производительность работы сервера. Кроме того, оптимизировано использование сервером системных ресурсов при построении и обновлении индексов, а также появилась возможность указать явным образом порядок размещения (по убыванию или по возрастанию) индексируемых данных.

Каскадные изменения. В предыдущих версиях при работе с внешними ключами ответственность за корректность ссылок возлагалась на пользователя. Теперь все необходимые корректировки ключей выполняются автоматически.

Интеграция в Web. Существовавшие до этого средства публикации данных в сетях Интернет и интрапет, реализуемые мастером Web Assistant Wizard и набором хранимых процедур, позволяли генерировать статические HTML-страницы. Теперь с помощью технологии XML пользователи сети могут непосредственно обращаться к хранимой на сервере информации. В окне обозревателя можно указать запрос, по которому будет получена информация из базы в формате XML. При этом можно указывать готовые запросы и применять различные шаблоны представления информации.

Для поддержки XML-технологий в SQL Server 2000 внесены некоторые

рые изменения в синтаксисе оператора запроса **SELECT** и, в частности, в его конструкцию **FOR**. Задавая в операторе **FOR** слово **XML**, пользователь инициирует преобразование результатов запроса в формат документа **XML**. Поддерживаются три режима преобразования: в плоский текст (режим **RAW**), в простое дерево (режим **AUTO**) и в сложное многоуровневое дерево с явным указанием зависимостей между элементами (режим **EXPLICIT**).

Надежность. В предыдущей версии продукта в распоряжении администратора БД имелись две основные технологии повышения устойчивости системы к сбоям и отказам оборудования сервера: резервный сервер (*standby server*) и кластеризация (*fail-over support*).

Первая из них предполагает установку в сети одного или нескольких дополнительных серверов, на которые по командам оператора будет периодически дублироваться информация с основного сервера. В случае выхода из строя основного сервера резервный сервер конфигурируется как основной, после чего пользователи могут продолжить свою работу уже с резервным сервером. Имя и сетевой адрес резервного сервера отличны от имени и адреса основного сервера. Пользователи должны заново запустить все задачи, выполняемые до этого на основном сервере. Автоматического переноса задач с резервного сервера на основной сервер не производится. После восстановления поврежденного сервера он снова становится основным, а временно исполняющий его функции сервер — резервным. Все изменения, произошедшие на резервном сервере за время ремонта основного, с помощью команд оператора переносятся на основной сервер.

Другая более совершенная технология позволяет сервер БД представить для пользователя как виртуальный сервер (кластер), хотя физически он строится на базе двух компьютеров (узлов), дублирующих друг друга в реальном времени. При выходе из строя одного из узлов все решаемые задачи продолжают выполняться на другом доступном для пользователя узле. Никаких изменений в настройках выполнять при этом не требуется. Для работы кластера обычно применяется внешний дисковый массив RAID, на котором хранятся общие данные узлов.

В MS SQL Server 2000 появилась возможность создавать кластеры из четырех узлов вместо двух, а также расширились возможности управления кластером при упрощении его администрирования.

Репликация. Под репликацией понимается совокупность механизмов, обеспечивающих отображение изменений данных на одном сервере, на другие серверы. Термин «издатель» (*publisher*) обозначает сервер, предоставляющий информацию из своих БД другим серверам, термин «дистрибутор» (*distributor*) обозначает промежуточный сервер, распространяющий информацию от издателя, а термин «подписчик» (*subscriber*) — принимающий информацию сервер.

В MS SQL Server 7 реализована репликация сведением (merge replication), а также технология «подписчиков незамедлительного обновления» (immediately update subscriber). Обе они позволяют вносить изменения в опубликованные данные со стороны подписчиков. Первая технология позволяет пользователям изменять данные даже при отсутствии соединения между дистрибутором (издателем) и подписчиком. Вторая технология требует наличия постоянного соединения между подписчиком и издателем.

MS SQL Server 2000 появилась технология отложенного обновления (Queue updating), которая позволяет выполнять изменения опубликованных данных на подписчике в случае отсутствия соединения его с дистрибутором (издателем). Она основана на ведении так называемой очереди изменений, данные из которойчитываются в момент появления соответствующего соединения. Кроме того, в распоряжении администраторов появилось более полутора десятка различных механизмов разрешения конфликтов изменения, возникающих при выполнении репликации сведений.

Новые типы данных. В распоряжении пользователей появилось три новых типа данных:

- **bigint**, который позволяет задавать целые числа в более широком диапазоне, и использует для представления числа 8 байт в отличие от обычного **int**, который использует 4 байта;
- **sql_variant**, позволяющий хранить значения большинства других типов MS SQL Server 2000 (числовые, символьные, денежные, даты и т. д.) и применяемый при работе с переменными, а также и со столбцами таблицы;
- **table**, позволяющий хранить сложные наборы данных наподобие таблиц. Его можно использовать для работы только с локальными переменными.

Масштабируемость. Как и ранее продукт обеспечивает высокий уровень масштабируемости, поскольку в зависимости от используемой редакции он может применяться пользователями как в домашних условиях в среде Windows 98, так и на многопроцессорных корпоративных серверах. Имеется возможность применения только ядра SQL Server 2000, что может быть использовано при создании независимыми разработчиками систем, требующих механизмов хранения и обработки данных.

В качестве сетевого сервера (в зависимости от масштаба организации, количества пользователей и нагрузки) могут использоваться различные редакции SQL Server 2000 — от Personal Edition, рассчитанной на Windows 98 и небольшое количество пользователей (до пяти), до Enterprise Edition, позволяющей использовать системы с большим количеством процессоров (до 32) и объемом оперативной памяти до 64 Гбайт, а также поддерживающей кластеры.

Службы *SQL Server 2000*

SQL Server 2000 реализован в виде служб операционной системы, что позволяет ему работать как часть ОС, иметь собственные права и не зависеть от работающего в данный момент пользователя. Это следующие четыре службы:

- MSSQLServer;
- SQLServerAgent;
- Microsoft Search (MSSearch);
- Microsoft Distributed Transaction Coordinator (MSDTC).

Поскольку Windows 98 не содержит служб, для работы SQL Server 2000 под ее управлением выполнена эмуляция работы служб. Перечислим функции каждой из служб.

Служба MSSQLServer является основной службой сервера, реализующей функции: регистрации пользователей и контроля их прав доступа; установления соединений; обслуживания обращений пользователей к БД; выполнения хранимых процедур; работы с файлами БД и журналом транзакций; контроля над использованием системных ресурсов и оперативной настройки сервера и многие другие. Все остальные службы можно рассматривать как расширение этой службы.

Основное назначение службы *SQLServerAgent* – автоматизация администрирования и использования SQL Server 2000. В ее задачи входит автоматический запуск заданий и извещение операторов о сбоях в работе сервера. Примером задания может быть автоматический запуск операций резервного копирования и проверки целостности базы данных во время наименьшей активности пользователей. Для запуска службы требуется предварительный запуск службы MSSQLServer. Большая часть функций службы реализована в виде хранимых процедур, выполняемых службой MSSQLServer. В работе службы используются объекты трех типов: задания (jobs), операторы (operators) и события (alerts). Для управления объектами можно использовать различные средства: утилиту SQL Server Enterprise Manager, операторы языка Transact SQL и программный интерфейс SQL-DMO.

Служба MSSearch, называемая также Full-Text Search (полнотекстовый поиск), используется для поиска символьной информации в таблицах баз данных сервера. Она существенно переработана по сравнению с предыдущей версией сервера. Пользуясь полнотекстовым поиском, можно не только находить слова и фразы, идентичные исковым, но и близкие к ним по смыслу и написанию. В результирующий набор включаются склоняемые формы глаголов и существительные. В работе службы используются специальные полнотекстовые каталоги (full-text catalog) и полнотекстовые индексы (full-text index), хранимые отдельно от основных данных, которые обновляются в процессе работы сервера. Администратор БД должен планировать интервалы времени обновления файлов полнотекстового поиска, а также выполнять их резервное копирование и восстановление.

Служба MSDTC используется для управления выполнением так называемых распределенных транзакций (*distributed transaction*). Необходимость в их вызове возникает в случаях одновременной работы с несколькими источниками данных, поддерживающими технологию OLE DB. В качестве таких источников могут выступать реляционные СУБД, текстовые файлы, книги MS Excel и настольные приложения. Распределенная транзакция реализуется как совокупность транзакций, открываемых на каждом источнике данных. Служба MSDTC синхронизирует эти транзакции, пользуясь двухфазным протоколом 2PC (*two-phase commit protocol*).

Кроме того, существует возможность запуска сервера как отдельного приложения. Для этого достаточно запустить на выполнение программу `sqlservr.exe`, находящуюся в папке `\Program Files\Microsoft SQL Server\MSSQL\Binn`.

Режимы работы сервера

MS SQL Server 2000 имеет две основные области применения: системы оперативной аналитической обработки и системы оперативной обработки транзакций. В соответствии с этим продукт имеет два режима работы. К сожалению, в сервере отсутствуют встроенные средства переключения в тот или иной режим, поэтому настройка требуемых параметров сервера администратором выполняется вручную.

Системы оперативной аналитической обработки или системы поддержки принятия решений (DSS, Decision Support System) ориентированы на представление пользователю высокопроизводительных и удобных средств многостороннего анализа данных. Эти системы имеют следующие особенности.

- Ориентированы на анализ данных, поэтому наиболее частой операцией является выборка данных, и практически не используются операции модификации данных. Сервер в таком режиме работы может выполнять запросы с большей скоростью, так как не нужны механизмы блокировок и транзакций. Кроме того, страницы данных могут заполняться на 100%, что позволяет более эффективно использовать внешнюю память и увеличить объем извлекаемой из базы информации за одно обращение к ней.
- Важнейшим показателем качества работы является скорость выполнения запросов, которая не должна превышать нескольких секунд. Для этого следует создавать необходимое количество индексов, а также использовать технологию материализованных представлений. Материализованное представление предназначено для предварительного выполнения конкретного запроса и изменения в него вносятся по мере изменений данных в базе. Поэтому при запуске запроса системе не требуется выполнять большое количество операций. Кроме того, для увеличения скорости выполнения запросов с использованием внешних источников дан-

ных сервер должен иметь средства прямого доступа к этим данным без промежуточного их переноса.

- Для быстрого выполнения анализа должны быть встроенные средства численного и статистического анализа информации.
- Всесторонний и глубокий анализ данных требует специального их представления, например, в виде многомерной модели (см. подраздел 2.5).
- Системы должны обеспечивать высокую безопасность, отвергая попытки несанкционированного доступа и предоставляя информацию авторизованным пользователям.

Полноценной реализацией OLAP фирмой Microsoft предлагается система MS DSS (Microsoft Decision Support System). Функции серверной части этой системы реализуются компонентом OLAP Services, поставляемым в составе SQL Server 2000, но устанавливаемым отдельно. Клиентская часть системы представлена инструментом PivotTable Service.

Системы оперативной обработки транзакций (OLTP, Online Transaction Processing) характеризуются большим количеством изменений в данных баз и одновременной работой множества пользователей с одними и теми же данными. Работа сервера в таком режиме требует активного использования механизмов транзакций и блокировок. Для обеспечения высокой скорости вставки новых данных, а также изменения данных, используемых в кластерном индексе, поддерживается низкая степень заполнения страниц. Это требует большего объема внешней памяти для хранения данных по сравнению с предыдущим режимом работы.

Инструменты SQL Server 2000

Инструменты администрирования могут устанавливаться как во время инсталляции самого сервера, так и отдельно. Это позволяет организовать рабочее место администратора на другом компьютере. Их можно использовать для управления любым сервером SQL Server 2000 или SQL Server 7.0 локальной сети.

Основными инструментами администрирования SQL Server 2000 являются следующие:

- SQL Server Enterprise Manager;
- SQL Server Service Manager;
- SQL Server Profiler;
- Query Analyzer;
- Upgrade Wizard (Мастер обновления);
- Import and Export Data (Мастер экспорта/импорта);
- утилиты Client Network Utility и Server Network Utility;
- утилиты командной строки;
- специальные Мастера (Wizards).

Большая часть административных задач SQL Server может быть выполнена тремя различными способами: с использованием средств Transact-SQL (требует высокой квалификации, но позволяет решать наиболее сложные задачи), с помощью графического интерфейса SQL Server Enterprise Manager (высокая функциональность наряду с простотой использования) и используя мастера (простота использования при ограниченном круге решаемых задач).

SQL Server Enterprise Manager является важнейшим инструментом, позволяющим выполнять следующие действия:

- управлять системой безопасности;
- создавать БД и ее объекты;
- создавать и восстанавливать резервные копии;
- конфигурировать подсистему репликации;
- управлять параметрами работы служб SQL Server 2000;
- управлять подсистемой автоматизации;
- запускать, останавливать и приостанавливать службы;
- конфигурировать связанные и удаленные серверы;
- создавать, управлять и выполнять пакеты службы трансформации данных DTS (Data Transformation Services).

SQL Server Service Manager предоставляет пользователю удобный механизм запуска, останова и приостановки служб SQL Server 2000. Кроме того, утилита позволяет разрешать или запрещать автоматический запуск той или иной службы при загрузке операционной системы.

SQL Server Profiler является графическим инструментом администратора для наблюдения за работой SQL Server 2000. При выполнении пользовательских запросов, хранимых процедур, подключения и отключения к серверу и других действий программы ядра сервера записывают в системные таблицы информацию о результатах выполнения этих действий. Утилита SQL Server Profiler с помощью специальных хранимых процедур выбирает эту информацию и представляет ее в удобном для анализа виде. Мониторинг сервера заключается в наблюдении за *событиями* (events), каждое из которых является минимальным контролируемым объемом работы ядра сервера. Каждое событие принадлежит какому-либо *классу событий* (event classes), которые в свою очередь для удобства объединены в двенадцать *категорий* (category).

Query Analyzer предназначен для выполнения запросов и анализа их результатов. По важности этот инструмент сопоставим с SQL Server Enterprise Manager. Так, в этой утилите появился обозреватель объектов (Object Browser), с помощью которого можно просмотреть список всех объектов, имеющихся в любой БД, а также перечень встроенных функций и системных типов данных.

Более существенным нововведением является возможность трассировки выполнения хранимых процедур. Для запуска трассировки достаточно в кон-

текстном меню необходимой процедуры выбрать пункт Debug. При выполнении трассировки можно указывать точки останова и выполнять хранимую процедуру в пошаговом режиме. Кроме того, с помощью Query Analyzer можно оценивать производительность исполнения запросов. При этом можно получать оценочный или прогнозируемый (estimated) и реальный (execution) планы выполнения запроса. Эти планы характеризуют затраты сервера на выполнение отдельных шагов запросов. Чтобы получить нужный план, достаточно из меню Query утилиты выбрать пункт Display Estimated Execution Plan или Show Execution Plan.

Мастер *Upgrade Wizard* используется для обновления баз данных SQL Server 6.5 до SQL Server 2000.

Мастер экспорта/импорта *Import and Export Data* предназначен для создания пакета DTS, который будет выполнять копирование информации между двумя источниками данных. Достоинством мастера является легкость выполнения с его помощью несложных задач.

Утилиты *Client Network Utility* и *Server Network Utility* предназначены для конфигурирования сетевых параметров на клиентской и серверной частях сервера соответственно. Сетевые параметры устанавливаются с помощью специальных сетевых библиотек, реализованных в виде динамически подключаемых к операционной системе библиотек (DLL, dynamic link library).

Сетевые утилиты Client (Server) Network Utility запускаются из главного меню операционной системы Programs\Microsoft SQL Server\Client(Server) Utility. Кроме того, первую утилиту можно вызвать путем запуска программы cliconfg.exe из каталога \WINNT\system32, а вторую — запуском программы svrnetcn.exe из каталога \Program Files\Microsoft SQL Server\80\Tools\Binn.

Утилиты командной строки, как правило, не имеют графического интерфейса и предназначены для выполнения различных задач, в том числе и функции ядра. Утилиты находятся в каталоге Binn установочного каталога SQL Server 2000. Примеры: console.exe (программа просмотра сообщений, выдаваемых сервером при создании и восстановлении резервных копий), dtsrun.exe (программа управления пакетами DTS), isqlw.exe (Query Analyzer), odbcping.exe (программа для проверки возможности установления соединения с сервером SQL Server 2000 с использованием ODBC), sqlserver.exe (программа реализации службы MSSQLServer).

Мастера предназначены для автоматизации и упрощения решения различных административных задач. Большинство мастеров обладает достаточно ограниченными возможностями. Однако, некоторые мастера, например, мастер конфигурирования подсистемы репликации таковым не является. Примеры мастеров: Backup Wizard (создание резервных копий базы данных), Create Database Wizard (создание БД), Create Diagram

Wizard (создание диаграммы БД), Create Index Wizard (создание индекса), Create Login Wizard (создание учетной записи SQL Server для пользователя), Create Stored Procedure Wizard (создание хранимой процедуры), Create View Wizard (создание представления), Index Tuning (оптимизация индексов).

Запустить мастер из Enterprise Manager можно с помощью пункта меню Tools | Wizards или путем нажатия кнопки Run a Wizard на панели инструментов.

Варианты поставки

SQL Server имеет следующие варианты поставки (редакций):

- Enterprise (Предприятий);
- Standard (Стандартный);
- Desktop Engine (Настольный);
- Windows CE (для работы в среде Microsoft Windows CE);
- Personal (Персональный);
- Developer (для разработки и тестирования приложений);
- Evaluation (Оценочный или Пробный).

SQL Server для своей установки предъявляет практически такие же требования к аппаратному обеспечению как и SQL Server 7.0, поскольку его ядро не подверглось большим изменениям. Эти требования можно считать достаточно высокими для обычного пользователя и вполне приемлемыми для промышленных серверов (см. табл. 13.1).

Таблица 13.1
Требования SQL Server 2000 к аппаратному обеспечению

Характеристика	Минимальные требования	Рекомендуемые требования
Процессор	Pentium 166, Pentium Pro	Pentium II или Pentium III
Оперативная память	32 Мбайт (64 для Enterprise Edition)	64–128 Мбайт (128–256 Мбайт для Enterprise Edition)
Дисковая память	65 Мбайт для минимальной установки; 90 Мбайт при установке только утилит администрирования; 170 Мбайт для типичной установки; 180 Мбайт для полной установки; +50 Мбайт для установки OLAP; +12 Мбайт для установки English Query	Аналогичны минимальным, но с учетом размеров пользовательских БД, а также с учетом возможного возрастания размеров системных БД

Возможно, что SQL Server 2000 будет работать и на более слабых компьютерах, но при этом не следует ожидать от него высокой производительности.

На каждую из существующих операционных систем семейства Windows возможна установка только определенных редакций SQL Server 2000 (см. табл. 13.2).

**Таблица 13.2
Соотношение редакций и операционных систем**

Операционная система	Enterprise Edition	Standard Edition	Personal Edition	Developer Edition	Desktop Engine	SQL Server CE	Enterprise Evaluation Edition
Windows 2000 Datacenter/ Advanced Server/Server	+	+	+	+	+	-	+
Windows 2000 Professional	-	-	+	+	+	-	+
Windows NT 4.0 Server/Enterprise Edition	+	+	+	+	+	-	+
Windows NT 4.0 Workstation	-	-	+	+	+	-	+
Windows 98	-	-	+	-	+	-	-
Windows CE	-	-	-	-	-	+	-

Работа SQL Server 2000 для более полных с функциональной точки зрения вариантов (Enterprise и Standard Edition) ведется *под управлением операционной системы Windows NT/2k*, размещенной на компьютере-сервере. Каждый пользователь получает доступ к ресурсам SQL Server с помощью персонального компьютера-клиента.

На *компьютере-клиенте* устанавливают одну из операционных систем Windows 9x, CE, DOS, OS/2 или Windows NT/2k. Эти операционные системы позволяют запускать различные приложения независимо от компьютера-сервера. Кроме того, каждый пользователь компьютера-клиента с помощью сетевых средств операционной системы может устанавливать связь с компьютером-сервером. На компьютерах-клиентах могут размещаться локальные базы данных, работа с которыми ведется с помощью персональных СУБД, например Access или Visual FoxPro. С их помощью осуществляется доступ к базам данных, размещенным на сервере.

Сеть, работающая под управлением операционной системы Windows NT/2k, строится в виде набора доменов, в каждом из которых может находиться

несколько рабочих групп. *Домен* (domain) представляет собой фрагмент сети Windows NT/2k. Перед добавлением к домену нового компьютера для него создается учетная запись на сервере. Права доступа владельца этого компьютера в учетной записи устанавливаются администратором.

Рабочая группа (workgroup) представляет собой логическое объединение нескольких компьютеров сети по некоторому признаку (отделам, задачам и т. п.). Объединение компьютеров пользователей в группы выполняется для облегчения совместного использования ресурсов.

В SQL Server 2000 наряду с рабочей группой используется понятие *роли* (role). Включая учетную запись в ту или иную роль сервера, можно предоставить ей определенный набор прав по администрированию сервера. Каждому пользователю может быть назначено произвольное число ролей. Например, пользователю может быть назначена роль администратора.

Кроме того, серверы SQL Server 2000 могут включаться в *серверные группы* (server groups), которые обеспечивают способ объединения большого числа серверов в несколько удобно управляемых групп.

Названные понятия имеют большое значение в обеспечении безопасности хранения и доступа к информации. Для совместного использования ресурсов в SQL Server возможно применение различных подходов. Роли ориентированы, прежде всего, на пользователей и в них не предусмотрены развитые средства обеспечения безопасности. Домены используются в Windows NT/2k, где имеются развитые средства безопасности и выполнения функций администрирования.

13.2. Язык запросов Transact-SQL

Для создания и работы с базами данных в SQL Server используется диалект языка SQL, именуемый Transact-SQL. По сравнению с первоначальным языком SQL, в Transact-SQL введены дополнительные ключевые слова, используемые при выборке, сохранении и выполнении операций над данными.

Основные операторы **INSERT**, **DELETE**, **UPDATE** и **SELECT** и другие операторы имеют общий синтаксис языка SQL для выполнения операций над данными. Дополнительные возможности Transact-SQL в основном связаны с управлением потоками информации и позволяют определять порядок выполнения операторов.

Операторы Transact-SQL можно задавать с помощью утилиты ISQL (Interactive Structured Query Language — интерактивный язык структурированных запросов), работающей под управлением MS DOS. Версия ISQL для Windows называется анализатором запросов (Query Analyzer).

В сеансе *MS DOS* утилиту *ISQL* можно вызвать с помощью одноименной команды. В командной строке запуска *ISQL* можно использовать параметры. Например, можно ввести имя пользователя и пароль:

```
ISQL /Usa /P<пароль> /S<сервер>
1>
```

Приглашения командной строки последовательно нумеруются автоматически до тех пор, пока не будет задана команда GO, являющаяся признаком завершения ввода команд и начала их выполнения.

Утилита *Query Analyzer* позволяет работать с операторами Transact-SQL под управлением Windows. При этом команды Transact-SQL вводятся в отдельном подокне запроса главного окна утилиты *Query Analyzer*. С помощью этой утилиты можно удобно копировать, вырезать, вставлять, редактировать, сохранять и печатать ранее созданные запросы.

После запуска *Query Analyzer* требуется подключиться к базе данных SQL Server с указанием имени пользователя, пароля (при необходимости) и используемого сервера. Операторы Transact-SQL вводятся в подокне *Query* (Запрос). Результаты выполнения запроса отображаются на вкладке *Results* (Результаты) в нижней части диалогового окна программы.

13.3. Системные базы данных и таблицы

База данных в SQL Server представляет собой логический объект, в котором размещаются таблицы и индексы. Физически база данных содержится в одном или нескольких файлах операционной системы. В предыдущих версиях SQL Server для размещения БД создавалось так называемое устройство, представлявшее логическое имя физического файла ОС.

Таблица (table) представляет собой набор полей и записей. Различают таблицы двух типов: *постоянные* и *временные*. Постоянные таблицы существуют до тех пор, пока их не удалят. Временные таблицы подразделяют на локальные и глобальные. *Локальные* временные таблицы существуют в текущем сеансе и затем уничтожаются. *Глобальные* временные таблицы существуют до завершения всех использующих их сеансов.

Журнал транзакций представляет собой рабочую область, в которую SQL Server записывает информацию до и после выполнения транзакций. Эта информация может использоваться для отмены выполненной транзакции или для восстановления БД. Журнал транзакций размещается в отдельном файле, создаваемом автоматически при создании базы данных.

Одной из важнейших возможностей SQL Server 2000 является высокая степень адаптации и огромные возможности настройки системы при изменении ее текущих параметров и условий функционирования. Так, при добавлении

данных файлы базы данных и журнала транзакций распределяются автоматически. Первичальный и максимальный размеры файлов, а также размер шага приращения указываются при создании базы данных.

Для хранения баз данных используются следующие три типа файлов:

- основной (Primary) файл создается один и содержит информацию, требуемую для инициализации. По умолчанию файл имеет расширение *mdf*;
- вспомогательные (Secondary) файлы содержат данные, не уменьшающиеся в основном файле; использование их не обязательно, но позволяет расположить БД на нескольких носителях. По умолчанию файлы имеют расширение *ndf*;
- файлы журналов транзакций хранят информацию для восстановления БД. По умолчанию файл имеет расширение *ldf*.

Кроме того, могут создаваться дополнительные группы файлов для размещения пользовательских данных.

Для хранения данных используются таблицы, размещаемые в базах данных. В Microsoft SQL Server базы данных делят на два типа — *системные* и *пользовательские*. В *системных* базах данных размещаются метаданные, используемые для управления системой. При инсталляции Microsoft SQL Server создаются следующие системные базы данных: *master*, *model*, *tempdb* и *msdb*.

Системная база данных *master* обеспечивает управление пользовательскими базами данных и работу Microsoft SQL Server. В ней содержатся следующие данные:

- учетные записи пользователей;
- сведения о текущих процессах;
- сообщения о системных ошибках;
- сведения о базах данных на сервере;
- выделенные размеры баз данных;
- сведения об активных блокировках;
- сведения о доступных и резервных устройствах баз данных;
- процедуры системного администрирования.

Ввиду важности этой базы данных рекомендуется иметь ее архив, отражающий самое последнее состояние.

Системная база данных *model* представляет собой шаблон для баз данных, создаваемых на текущем сервере. Она содержит системные таблицы, необходимые каждой пользовательской базе данных. В базу данных *model* помещают объекты, которые должны присутствовать в создаваемых базах данных. Обычно такими объектами являются следующие:

- типы данных, определяемые пользователями;
- правила проверки ввода;

- значения по умолчанию;
- хранимые процедуры;
- информация о пользователях, которым разрешается доступ к базам данных;
- разрешения, записываемые по умолчанию в учетные записи гостей.

Системная база данных **tempdb** служит для размещения на диске различных временных объектов: таблиц, промежуточных результатов предложений группирования и упорядочения, курсоров и др.

Содержимое базы данных *tempdb*, кроме глобальных временных таблиц, удаляется при разрыве пользователем соединения с SQL Server. При завершении работы с SQL Server из этой базы удаляются все данные. В базе данных *tempdb* размещаются временные таблицы всех БД, с которыми ведется работа.

Системная база данных **msdb** используется для обеспечения работы службы SQLServerAgent. В ней хранится информация, относящаяся к автоматизации администрирования и управления SQL Server 2000, информация об операторах и событиях, а также информация о расписании автоматического запуска заданий.

Кроме системных таблиц, образующих каталог баз данных (database catalog), в базе данных *msdb* содержатся следующие системные таблицы: *sysalerts*, *sysoperators*, *sysnotifications*, *sysjobhistory*, *sysjobs* и другие. Эти таблицы соответственно содержат информацию: об определенных пользователем событиях; об операторах и извещениях; о выполнении каждого из шагов задания (истории); всех заданиях, сконфигурированных на сервере и т.д.

Каталог баз данных (database catalog) представляет собой совокупность около 20 системных таблиц, которые имеются в каждой базе данных. Имена всех системных таблиц начинаются с приставки sys, например: *sysalternates*, *syscolumns*, *syscomments*. В них хранится информация о пользователях, триггерах и хранимых процедурах, таблицах, индексах в таблицах, правах доступа пользователей, типах данных, ограничениях, публикациях, репликациях и другая. Наименования системных таблиц mnemonicски указывают на характер хранимой информации. Например, в системной таблице *sysindexes* содержатся данные об индексах, таблицах без индексов.

Системный каталог, или **словарь данных**, представляет собой некоторую совокупность системных таблиц, размещенных в базе данных **master**. В состав системного каталога входят следующие системные таблицы: *syscharsets*, *sysconfigures*, *syscurconfigs*, *sysdatabases*, *sysdevices*, *syslanguages*, *syslocks*, *syslockinfo*, *sysxlogins*, *sysmessages*, *sysprocesses*, *sysservers* и другие.

Заметим, что информация о том, какие объекты находятся в базе данных, размещается в системной таблице *sysobjects*, имеющейся в каждой базе данных.

13.4. Создание баз данных

При работе с базами данных, размещеными на сервере, можно выделить следующие этапы:

- создание базы данных и таблиц;
- создание представлений и хранимых процедур;
- работа с таблицами;
- восстановление данных;
- администрирование.

Основными средствами реализации названных этапов в среде Microsoft SQL Server 2000 являются SQL Server Enterprise Manager, а также язык Transact-SQL.

Создание базы данных может быть выполнено пользователем при наличии у него соответствующих полномочий, устанавливаемых системным администратором. Для создания базы данных можно использовать SQL Server Enterprise Manager или команду Transact-SQL `CREATE DATABASE`. Напомним, что при создании базы данных в качестве шаблона используется база данных *model*, содержащая ряд системных таблиц.

Для создания базы данных с помощью SQL Server Enterprise Manager необходимо следующее.

1. Запуск названной программы, выбор в диалоговом окне сервера баз данных и выделение папки Databases (рис. 13.1).
2. Выполнение команды Action | New Database (Действие | Создать базу данных).

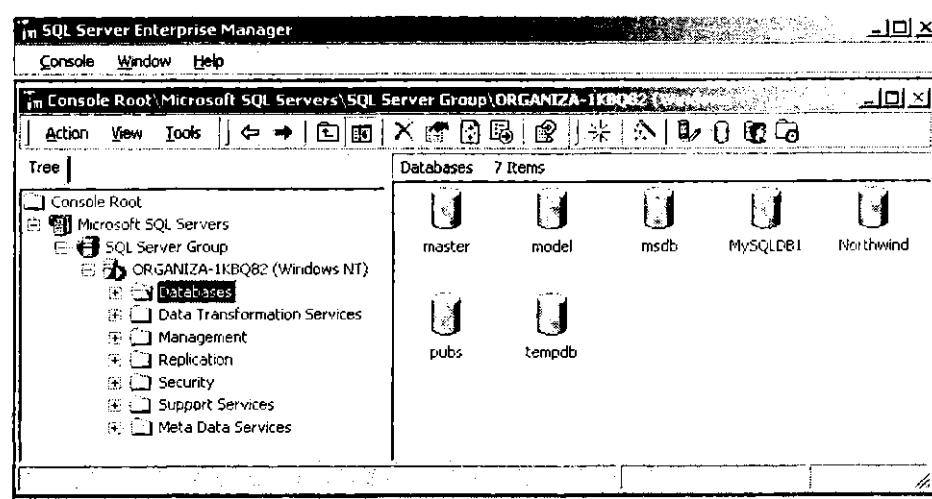


Рис. 13.1. Диалоговое окно SQL Server Enterprise Manager

3. В открывшемся диалоговом окне свойств задание параметров базы данных: имени, местоположения файлов, первоначального размера, возможности автоматического увеличения размера.

Для создания базы данных предназначена команда Transact-SQL следующего формата:

```
CREATE DATABASE имя БД
[ON [ <спецификация файла> [...п] ], <описание группы файлов> [...п]]
[LOG ON <спецификация файла> [...п]]
[COLLATE имя сопоставления]
[FOR LOAD | FOR ATTACH]
<спецификация файла> ::= [PRIMARY]
    (NAME = логическое имя файла ,
     FILENAME = ' имя файла операционной системы '
        [, SIZE = размер ]
        [, MAXSIZE = { максимальный размер | UNLIMITED}]
        [, FILEGROWTH = приращение ])
<описание группы файлов> ::= FILEGROUP <имя группы файлов> <спе-
цификация файла> [...п]
```

Параметры команды CREATE DATABASE имеют следующий смысл:

- ON определяет список файлов на диске для размещения информации базы данных;
- п — на месте этого символа можно указать дополнительные файлы для размещения БД;
- LOG ON определяет список файлов на диске для размещения журнала транзакций;
- COLLATE указывает сопоставление, которое будет иметь БД;
- OR LOAD используется для совместимости с предыдущими версиями SQL Server;
- FOR ATTACH задает не создание новой БД, а выполнение присоединения существующей БД;
- COLLATE указывает сопоставление, которое будет иметь БД;
- PRIMARY определяет первый файл. Если параметр опущен, то первичным является первый файл в списке;
- SIZE определяет первоначальный размер файла, минимальное значение параметра 512 Кбайтов, если он не указан по умолчанию принимается 1 Мбайт;
- MAXSIZE определяет максимальный размер файла базы данных. При значении параметра UNLIMITED максимальный размер базы данных ограничивается свободным местом на диске.

Перед созданием базы данных нужно открыть и установить текущей базу данных *master* с помощью команды USE master.

Пример 1. Создание базы данных.

```
CREATE DATABASE owndbase266
ON PRIMARY
(NAME = owndbase266_data,
FILENAME = 'C:\mssql11\data\owndbase266_data.mdf',
SIZE = 4 MB,
FILEGROWTH = 1 MB)
LOG ON
(NAME = owndbase266_log,
FILENAME = 'C:\mssql11\data\owndbase266_log.ldf',
SIZE = 2 MB,
FILEGROWTH = 1 MB)
```

Напомним, что при создании БД используется шаблон – база данных *model*, которая может быть модифицирована как любая другая база данных. Это означает возможность заранее создать требуемые объекты БД, которые будут входить в каждую создаваемую базу данных.

13.5. Работа с таблицами

При работе с таблицами предполагается создание таблиц, добавление данных в таблицу, выборка, удаление и изменение данных в таблице.

Создание таблиц в SQL Server 2000 можно выполнить либо с помощью графического интерфейса Enterprise Manager, либо используя команду CREATE TABLE языка Transact-SQL. Мастера создания таблиц в SQL Server 2000, к сожалению, нет.

Для создания *локальной временной* таблицы в ее имени первым указывают символ #. По окончании сеанса, в котором временная локальная таблица была создана, она автоматически удаляется. Имя локальной временной таблицы вместе с символом # может иметь не более 116 символов.

Для создания *глобальной временной* таблицы, доступной в любом сеансе, в ее имени первыми указывают два символа #. Сеансы создаются пользователями, работающими на компьютерах-клиентах. Глобальная временная таблица удаляется автоматически по окончании последнего сеанса, где она использовалась.

Целесообразность создания и использования временных таблиц может быть связана с объединением данных из нескольких таблиц и работой с ними

в течение сеанса. При этом доступ к объединенным данным естественно осуществляется быстрее, чем к данным из нескольких постоянных таблиц.

При создании таблицы для каждого ее поля задается **тип данных**, определяющий тип информации, которую можно хранить в поле. После определения типа данных столбца он сохраняется как постоянная характеристика. Как и в предыдущей версии, в SQL Server 2000 можно изменять структуру таблицы, например типы данных полей, с помощью оператора ALTER TABLE. Типы данных (системные) таблиц SQL Server 2000 приведены в табл. 13.3.

Таблица 13.3
Типы данных

Тип	Размер, байтов	Краткое описание
binary(n)	до 8 Кбайт	Двоичные данные фиксированной длины
varbinary(n)	до 8 Кбайт	Двоичные данные переменной длины
char(n)	до 8 Кбайт	Символьная строка фиксированной длины
nchar(n)	до 8 Кбайт	Символьная строка Unicode фиксированной длины
varchar(n)	до 8 Кбайт	Символьная строка переменной длины
nvarchar(n)	до 4 Кбайт	Символьная строка Unicode переменной длины
datetime	8	Дата и время высокой точности
smalldatetime	4	Дата и время низкой точности
decimal(p,s)	1–17	Число с общим количеством цифр p и числом s цифр после запятой
numeric(p,s)	1–17	Число с общим количеством цифр p и числом s цифр после запятой
float	8	Число с плавающей точкой
real	4	Число с плавающей точкой
bigint	8	Целое число
int	4	Целое число
smallint	2	Целое число
tinyint	1	Целое положительное число от 0 до 255
money	8	Денежное значение
smallmoney	4	Денежное значение
bit	1	Булево значение (бит, имеющий значение 0 или 1)
timestamp	8	Временная отметка или уникальный в рамках БД номер строки после ее обновления

Таблица 13.3 (продолжение)

Тип	Размер, байтов	Краткое описание
text	до 2 Гбайт	Текстовые данные
ntext	до 1 Гбайт	Текстовые данные Unicode
sql_variant		Хранит значения различных других типов, кроме text, ntext, timestamp, image и sql_variant
image	до 2 Гбайт	Двоичные данные (файлы MS Office, изображения и т. д.)
table		хранение результатов выполнения запросов
uniqueidentifier		Глобальный уникальный идентификатор (globally unique identifier, GUID)

Кроме приведенных системных типов данных таблиц, можно определять на их основе свои пользовательские типы данных для конкретной базы данных.

Для создания *постоянной* таблицы с помощью SQL Server Enterprise Manager нужно выполнить следующее:

- запустить SQL Server Enterprise Manager;
- в открывшемся диалоговом окне выбрать серверную группу щелчком по знаку «+» слева от имени группы;
- выбрать базу данных (рис. 13.2), в которой предполагается создание таблицы;

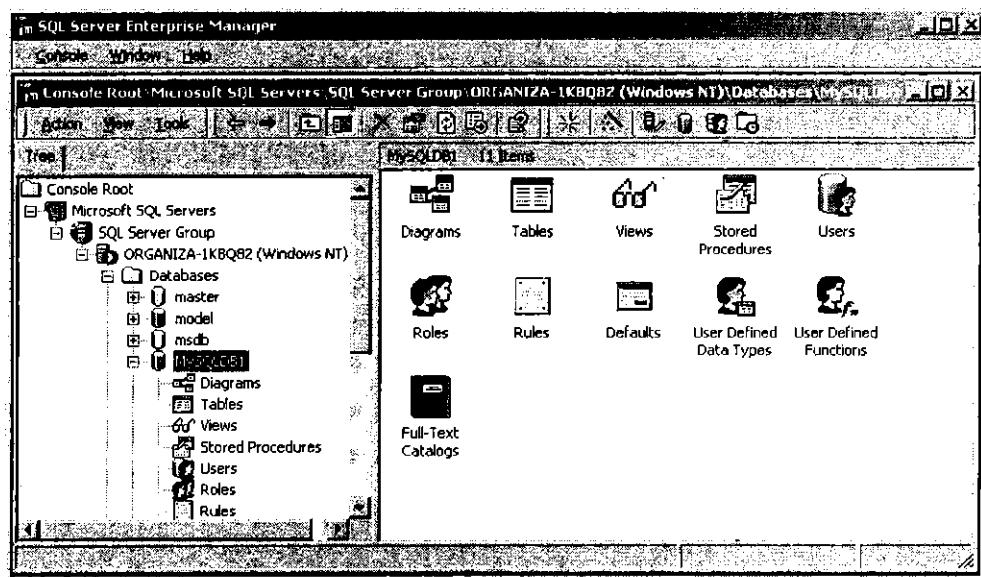


Рис. 13.2. Вид окна с объектами базы данных

- выбрать объект **Tables** (таблицы) базы данных и из контекстного меню или из пункта **Action** (действие) главного меню выбрать команду **New Table** (Новая таблица);
- в открывшемся диалоговом окне указать имена столбцов таблицы, определив для каждого столбца тип данных, при необходимости размер, допустимость значения **Null** и значение по умолчанию (**Default**);
- сохранить таблицу (кнопка **Save Table**), указав ее имя в окне **Specify Table Name**.

Создание таблицы можно выполнить с помощью команды **CREATE TABLE** языка Transact-SQL. Упрощенный формат команды выглядит следующим образом.

```
CREATE TABLE [имя_БД.[владелец.] | владелец.] имя_таблицы
( {<определение_столбца>} | имя_столбца AS вычисляемое_выражение |
<ограничения_таблицы> [,..n])
[ON {группа_файлов | DEFAULT}]
[TEXTIMAGE_ON {группа_файлов | DEFAULT}]

<определение_столбца> ::= имя_столбца тип_данных
[COLLATE <имя_сопоставления>]
[ [DEFAULT выражение] |
[IDENTITY [(начальное_значение, приращение)
[NOT FOR REPLICATION]]]
]
[ROWGUIDCOL]
[<ограничения_столбца>]
```

При создании таблицы можно указать базу данных и владельца, которым она принадлежит. Удобно предварительно определить текущую базу данных с помощью команды **USE база_данных**. При этом все последующие команды выполняются для этой базы данных, и ее имя можно не указывать. Имя таблицы и столбца таблицы не может иметь более 128 символов.

Наиболее часто структура таблицы задается перечислением полей (столбцов) и типов данных, которые в них хранятся. При этом могут также указываться другие необязательные параметры.

В частности, параметр с ключевым словом **COLLATE** задает используемое для данных столбца сопоставление.

Стандартное значение данных столбца можно указать с помощью параметра **DEFAULT <значение по умолчанию>**. Это значение будет автоматически появляться в столбце всякий раз при переходе пользователя к созданию новой строки.

Атрибут **IDENTITY** позволяет создать числовое поле с автоматическим приращением значений. Начальное значение, заданное для этого атрибута, автоматически помещается в первую строку этого столбца, а при добавлении в таблицу новых строк в этот столбец значения вводятся автоматически путем добавления приращения к значению из последней строки. Это свойство нельзя задавать для столбцов с атрибутом **NULL**. Если при задании свойства **IDENTITY** не указать начальное значение и приращение, то по умолчанию они примут значение 1.

NOT FOR REPLICATION означает отмену действия ограничений целостности для поля во время выполнения операций репликации (эта установка включена по умолчанию).

Если при описании столбца указать описатель **ROWGUIDCOL**, то этот столбец будет играть роль *столбца глобального уникального идентификатора строки* (*row global unique identifier column*). Такие столбцы необходимы для выполнения специальных операций, например, при репликации свидетельств.

Параметр **ON {группа_файлов | DEFAULT}** позволяет явно определить группу файлов, в которой необходимо создать таблицу. Эта группа файлов должна существовать в базе данных. Если параметр опущен или задано **DEFAULT**, то таблица будет создана в группе файлов, определенной как группа файлов по умолчанию.

Параметр **TEXTIMAGE_ON {группа_файлов | DEFAULT}** используется для указания отдельной группы файлов, используемой для хранения значений типа **text**, **ntext** и **image**. Это делается для повышения производительности обработки таблиц, имеющих указанные типы данных. Если параметр опущен или указано значение **DEFAULT**, то значения упомянутых типов данных будут размещены в группе файлов по умолчанию.

Параметры, задаваемые в конструкции **<ограничения_столбца>**, определяют ограничения целостности, действующие на уровне столбца. В составе этих ограничений могут быть следующие: определение первичного ключа (**PRIMARY KEY**), поддержка уникальности значений поля (**UNIQUE**), внешний ключ (**FOREIGN KEY**), возможность хранения неопределенных значений (**NULL** | **NOT NULL**), проверка истинности логического выражения для значений поля в операциях вставки и изменения (**CHECK**) и другие.

Параметры, задаваемые в конструкции **<ограничения_таблицы>**, определяют ограничения целостности, действующие на уровне таблицы. Эти параметры могут применяться в тех случаях, когда необходимо определить ограничение целостности на основе нескольких столбцов. Для определения параметров здесь во многом используются те же ключевые слова, которые служат для задания ограничений, действующих на уровне столбца.

Одним из специфических и, на наш взгляд, важных параметров ограничений целостности, действующих на уровне таблицы, является параметр опре-

деления порядка сортировки значений в таблице. Он выглядит следующим образом:

(имя_столбца {ASC | DESC} ...),

где ключевые слова **ASC** и **DESC** определяют порядок сортировки значений по возрастанию и по убыванию соответственно для каждого из указанных столбцов.

Пример 1. Создание таблицы.

Рассмотрим оператор создания таблицы с тремя столбцами целочисленных типов.

```
CREATE TABLE table1  
(int1 TINYINT, int2 INT, int3 SMALLINT)
```

Пример 2. Пусть требуется создать таблицу сотрудников, содержащую семь столбцов, используемых для хранения соответственно: цифрового кода сотрудника, фамилии, имени, отчества, пола (М – мужчина, Ж – женщина), даты рождения и должности. При этом код сотрудника должен играть роль первичного ключа таблицы, имя и фамилия не должны быть пустыми и данные таблицы должны сортироваться в алфавитном порядке фамилий и отчеств, а также в порядке увеличения возраста.

Оператор создания описанной таблицы может выглядеть следующим образом:

```
CREATE TABLE tab2  
(emp_id int IDENTITY (1,1) PRIMARY KEY,  
second_name varchar(30) NOT NULL,  
first_name varchar(15) NOT NULL,  
middle_name varchar(20) NULL,  
sex char(1),  
birth_date datetime,  
position varchar (50)  
CONSTRAINT const_tab2 UNIQUE  
(second_name ASC,first_name ASC,birth_date DESC)  
)
```

Свойство **IDENTITY** позволяет определить для столбца таблицы начальное значение, которое автоматически помещается в столбец в первую строку, и приращение, автоматически добавляемое к последнему значению введенному в столбец текущей строки. При добавлении в таблицу новых строк в

этот столбец значения вводятся автоматически путем добавления приращения к значению из последней строки.

Основные действия **по работе с данными** таблиц заключаются в выборке, добавлении, удалении и изменении информации.

Добавление данных в таблицу можно выполнить с помощью оператора INSERT следующего формата

```
INSERT INTO имя_таблицы  
(столбец_1,...,столбец_n)  
VALUES ('строка_1',...,'строка_n')
```

Во второй строке оператора перечисляются имена столбцов таблицы. В следующей строке за ключевым словом **VALUES** в круглых скобках указываются добавляемые в столбцы таблицы значения. Не обязательно перечислять имена столбцов в порядке их следования в таблице. Важно, чтобы порядок следования имен столбцов во второй строке оператора соответствовал порядку следования значений в третьей строке.

Каждый оператор **INSERT** позволяет добавить в таблицу одну запись (строку).

Пример 3. Добавление записи в таблицу.

Рассмотрим для трех столбцов добавление в таблицу строки из трех целочисленных значений.

```
INSERT INTO table1  
(int1, int2, int3)  
VALUES (255, 32767, 50000)
```

Пример 4. Добавление записи в таблицу.

```
INSERT INTO table2  
(Name, Sity, Year)  
VALUES ('Salivan', 'New York', 2004)
```

Атрибуты NULL И NOT NULL. Задание для столбца таблицы атрибута **NULL** позволяет опустить при вводе данных значения этого столбца. В случае задания атрибута **NOT NULL** SQL Server 2000 не позволяет оставить этот столбец пустым при вставке строки в таблицу. По умолчанию столбцу назначается атрибут **NOT NULL**.

Определенное для столбца значение **NULL** отличается от пробела, нуля и ASCII-символа нуля. Значение **NULL** для столбца интерпретируется как неопределенное или не заданное, поскольку при вставке строки в таблицу соответствующее значение не вводилось. При ссылке на строку в месте значения

столбца, содержащего **NULL**, отображается элемент **NULL**, указывающий на то, что в этом месте строки значение не вводилось.

Если при добавлении записи в таблицу пропустить одно или несколько имен столбцов, то в них помещается стандартное значение или **NULL**. Правила неявного присвоения значения столбцу, если в операторе **INSERT** значение столбца не указано, приведены в табл. 13.4.

Таблица 13.4.

Правила неявного присвоения значения столбцу

Определение столбца	Нет ввода значений		Ввод значения NULL
	Атрибута DEFAULT нет	Атрибут DEFAULT есть	
Атрибут NULL	NULL	Стандартное значение	NULL
Атрибут NOT NULL	Ошибка	Стандартное значение	Ошибка

Для вставки значения **NULL** в столбец, для которого задан атрибут **NULL**, следует в списке за ключевым словом **VALUES** для этого столбца указать **NULL**.

Для **выборки данных** из таблиц используется оператор **SELECT** языка SQL. С его помощью можно создавать разнообразные запросы на отбор информации из таблиц, позволяя указывать столбцы и строки. Для подготовки SQL-запроса на выборку данных из таблицы можно использовать следующие средства:

- утилиту *Query Analyzer*;
- утилиту командной строки *ISQL*;
- утилиту командной строки *OSQL*.

Отличие *OSQL* от *ISQL* заключается в том, что в *OSQL* для подключения к базе данных используется механизм ODBC, а не механизм DB-LIB. Обе утилиты обладают одинаковым синтаксисом, но вызываются со своими именами.

Наиболее удобно использование программы *Query Analyzer*, обладающей графическим интерфейсом пользователя (GUI – Graphic User Interface) и обеспечивающей удобную корректировку запросов. Кроме того, этой программой можно пользоваться непосредственно из программы *SQL Server Enterprise Manager*, которая облегчает работу с табличными структурами, другими серверами и другими возможностями *SQL Server 2000*.

Для запуска и использования утилиты *Query Analyzer* из программы *SQL Server Enterprise Manager* выполняются следующие действия.

1. Выбор в окне программы *SQL Server Enterprise Manager* сервера баз данных из списка зарегистрированных серверов.

2. Открытие окна Анализатора запросов с помощью команды Tools | SQL Query Analyzer (Сервис | Анализатор SQL запросов).
3. Выбор в диалоговом окне из списка DB базы данных, если она не выбрана ранее.
4. Ввод оператора запроса в верхней половине окна программы и нажатие клавиши <F5> или выбор команды Query | Execute (Запрос | Выполнить). Для отображения результатов запроса с линиями сетки необходимо после ввода оператора выбрать команду Query | Results In Grid и нажать клавишу <F5>.

Результаты запроса автоматически отображаются в нижней части окна программы Query Analyzer по мере получения информации.

13.6. Индексы и ключи

Индексы служат для задания нужного порядка вывода данных и ускорения выборки данных. При отсутствии индекса SQL Server для выполнения запроса должен отсканировать таблицу, т. е. просмотреть все строки таблицы. В случае большого размера таблиц на сервер ложится значительная нагрузка. Индексы играют роль указателей на нужные данные, позволяющих ускорить выборку данных из таблиц.

В базе данных индекс представляет собой список значений со ссылками на страницы таблиц, в которых содержатся нужные данные. С другой стороны, индексы представляют собой объекты базы данных, для размещения которых требуется дополнительная память.

При создании индекса SQL Server 2000 выполняет сканирование таблицы, выбирает значения в индексируемом столбце и записывает на индексную страницу указатели на страницы данных и идентификаторы строк для индексируемых значений.

В SQL Server 2000 допускаются следующие два типа индексов: кластерные и некластерные.

Для индексов **кластерного** типа порядок следования строк в индексе совпадает с физическим порядком расположения данных. Кластерный индекс непосредственно содержит данные, поэтому в таблице может быть один индекс кластерного типа. Такой индекс обеспечивает наибольшую скорость доступа к данным. Для задания кластерного индекса обычно выбирают столбец, который наиболее часто используется в операциях поиска, упорядочения или группирования. Например, для этого часто используется столбец, по которому построен первичный ключ.

Индексы **некластерного** типа имеют логическую связь с данными, поэтому для одной таблицы можно задавать несколько таких индексов.

Индекс создают путем указания имен столбцов, составляющих этот

индекс. Причем при выборке данных из таблиц используется следующий принцип: заданный для таблицы индекс используется, если один из столбцов, указанных в операторе **SELECT**, следует первым в индексе.

При выборе столбца для создания индекса нужно учитывать следующее:

- нельзя создавать индекс для столбцов, имеющих тип данных **BIT**, **TEXT** или **IMAGE**;
- нельзя создавать индекс для столбцов, имеющих тип данных **CHAR**, **VARCHAR**, **NCHAR**, **NVARCHAR**, **BINARY** и **VARBINARY**;
- в качестве основного кандидата для создания индекса целесообразно использовать первичный ключ;
- для таблиц, к которым строятся запросы, возвращающие большие результатирующие множества, целесообразно индексировать все, что возможно;
- целесообразно использовать в индексе столбцы, содержащиеся в директивах **ORDER BY** (упорядочение по) и **GROUP BY** (сортировка по).

В SQL Server 2000 можно использовать в одном запросе несколько индексов таблицы. При этом у разработчика приложения возникает вопрос, что лучше: использовать составные индексы или несколько простых индексов. Рекомендуется составной индекс создавать при необходимости выборки уникальных данных из таблиц, состоящих из большого множества строк. В остальных случаях лучше создавать множество простых индексов, включающих один столбец.

Индексы можно создать двумя способами: с помощью программы SQL Server Enterprise Manager и с помощью оператора **CREATE INDEX** языка Transact SQL. Создать индекс для таблицы может только ее владелец.

Для создания индекса с помощью программы SQL Server Enterprise Manager выполняется следующее:

1. Запуск названной программы из группы Microsoft SQL Server.
2. Выбор сервера, базы данных и таблицы, для которой выполняется создание индекса.
3. Выполнение команды меню **Action | All tasks | Manage Indexes** (Действие | Все задачи | Работа с индексами).
4. В открывшемся диалоговом окне **Manage Indexes** (Работа с индексами) нажатие кнопки **New** (Создать).
5. В очередном диалоговом окне **Create New Index** (Создание нового индекса) в поле **Index Name** (Имя индекса) задание имени индекса.
6. В группе **Index Options** (Параметры индекса) задание параметров индекса с помощью флажков и нажатие **OK**.

Выбор кластерного индекса определяется в диалоговом окне **Create New Index** (Создание нового индекса) в группе **Index Options** (Параметры индекса) установкой одноименного флажка **Clustered Index** (Кластерный индекс).

Можно указать явно для SQL Server, какой индекс следует использовать путем задания его в директиве **FROM**. С этой целью в операторе **SELECT** используется директива **INDEX**. Упрощенно формат оператора выборки данных можно представить следующим образом:

```
SELECT...
FROM имя_таблицы (INDEX = n)
```

Здесь ключевое слово **INDEX** предписывает, чтобы SQL Server использовал индекс, определяемый значением п. Если значение п равно нулю, то выполняется сканирование таблицы. При единичном значении п используется кластерный индекс, если он имеется в таблице. Остальные значения п определяют порядковые номера индексов, которые следует использовать.

В SQL Server имеется два способа *отображения* информации об индексах: с помощью диспетчера индексов Index Manager программы SQL Server Enterprise Manager, а также с помощью системных хранимых процедур **sp_helpindex** или **sp_statistics**.

Индексы можно *удалять* двумя способами: с помощью программы SQL Server Enterprise Manager и с помощью оператора **DROP INDEX** языка Transact SQL. Для *удаления* индекса с помощью программы SQL Server Enterprise Manager выполняется следующее:

1. Запуск программы из группы Microsoft SQL Server.
2. Выбор сервера, базы данных и таблицы.
3. Выполнение команды меню **Action | All tasks | Manage Indexes** (**Действие | Все задачи | Работа с индексами**).
4. В открывшемся диалоговом окне **Manage Indexes** (**Работа с индексами**) выбор удаляемого индекса и нажатие кнопки **Delete** (**Удалить**).
5. В очередном окне подтверждение необходимости удаления индекса или отказ.

В SQL Server **ключи** служат для поддержания целостности данных в таблицах баз данных (см. подразделы 3.1–3.2). Напомним, что *первичный* ключ представляет собой уникальное поле или набор полей, однозначно определяющий строки таблицы базы данных. *Внешний ключ* представляет собой поле или набор полей таблицы, который соответствует первичному ключу другой таблицы, связанной с исходной таблицей. Область значений для внешнего ключа является подмножеством значений соответствующего первичного ключа.

В SQL Server первичные и внешние ключи можно создать следующими тремя способами: с помощью программы SQL Server Enterprise Manager, с помощью оператора **ALTER TABLE...ADD CONSTRAINT** языка

Transact SQL и с помощью параметра PRIMARY/FOREIGN KEY оператора CREATE TABLE.

Для добавления *первичного* ключа с помощью программы SQL Server Enterprise Manager выполняется следующее:

1. Запуск программы из группы Microsoft SQL Server.
2. Выбор сервера, базы данных и таблицы.
3. Выполнение команды Action | Design Table (Действие | Конструктор таблиц).
4. Выделение столбцов, определяемых в качестве первичных индексов, и выполнение команды контекстного меню таблицы Set Primary Key (Установить первичный ключ).
5. Сохранение изменений в таблице с помощью щелчка на кнопке Save (Сохранить) панели инструментов (с изображением дискеты).

В SQL Server имеется два способа *отображения* информации о ключах: с помощью диспетчера таблиц Table Manager программы SQL Server Enterprise Manager и системных хранимых процедур sp_help и sp_helpconstraint, а также с помощью хранимых процедур ODBC sp_keys и sp_fkeys.

Первичные и внешние ключи можно *удалить* с помощью программы SQL Server Enterprise Manager и с помощью оператора ALTER TABLE...DROP CONSTRAINT языка Transact SQL.

13.7. Хранимые процедуры и триггеры

Хранимые процедуры представляют собой фрагменты программного кода на Transact-SQL, которые выполняются на сервере. Они могут запускаться вызывающим их приложением, правилами проверки целостности данных или триггерами. Хранимые процедуры могут иметь параметры, обеспечивающие возможность передавать в нее значения и получать обратно значения, выбираемые из таблиц или вычисляемые при выполнении процедуры.

Основным достоинством использования хранимых процедур является высокая оперативность обработки информации, обусловленная использованием, как правило, мощного компьютера-сервера и тем, что для доступа к данным не тратится лишнее время, поскольку база данных размещается на сервере.

В сравнении с использованием динамических операторов языка SQL хранимые процедуры отличают следующие преимущества: высокая производительность, поскольку они компилируются при первом выполнении с использованием оптимизации доступа к информации из таблиц; возможность выполнения на локальном и на удаленном сервере.

По области видимости различают хранимые процедуры следующих четырех типов: системные (System), локальные (Local), временные (Temporatge) и удаленные (Remote).

Системные хранимые процедуры размещаются в базе данных *Master*, используются сервером и администратором. Имена системных процедур начинаются с символов *sp_*. Для создания собственной системной хранимой процедуры достаточно присвоить ей имя, начинающееся с *sp_* и поместить ее в базу данных *Master*.

Локальные хранимые процедуры размещаются в пользовательских базах данных.

Временные хранимые процедуры имеют имена, начинающиеся с символов *#* или *##*. Процедуры с одним символом *#* в начале имени являются временными *локальными* процедурами и доступны из соединения, в котором они были созданы. При закрытии этого соединения процедура автоматически уничтожается. Процедуры с двумя символами *##* в начале имени являются временными *глобальными* и доступны в любой сессии работы с данным сервером.

Удаленные хранимые процедуры можно вызывать с сервера, отличного от текущего сервера. При этом текущий сервер играет роль промежуточного звена. Кроме того, удаленную хранимую процедуру можно вызвать непосредственно с сервера, на котором она находится.

Еще одним типом хранимых процедур являются так называемые *расширенные хранимые процедуры* (extended stored procedures). Они пишутся на языке программирования, таком как С. Расширенные хранимые процедуры оформляются как функции в составе библиотек динамических связей — DLL (Dynamic Link Library), что повышает скорость их выполнения. Имена расширенных хранимых процедур начинаются с символов *xp_*.

Хранимые процедуры можно создавать как последовательности операторов Transact-SQL (см. одноименный подраздел) с использованием оператора **CREATE PROCEDURE** и с помощью программы SQL Server Enterprise Manager.

Имя хранимой процедуры может иметь до 128 символов. В хранимой процедуре могут использоваться все операторы SQL, кроме **CREATE**. По умолчанию разрешение на выполнение хранимой процедуры получает владелец базы данных, который может предоставлять эти права другим пользователям. Оператор *создания хранимой процедуры* имеет следующий формат:

```
CREATE PROCEDURE [владелец.]имя_процедуры [;версия]
[(@имя_параметра тип [=default] [OUTPUT]
...
[@имя_параметра тип [=default] [OUTPUT]])]
[FOR REPLICATION | WITH RECOMPILE][, ENCRYPTION]
AS
<операторы_Transact-SQL>
Использование в имени процедуры составляющей «;версия» (задаваемой
```

целыми числами) позволяет получить группу одноименных процедур с одинаковыми именами и разными версиями, например: proc;1, proc;2, proc;3 и т. д. Это удобно тем, что всю такую группу процедур можно удалить одной следующей командой

```
DROP PROCEDURE proc
```

Вызов хранимой процедуры для выполнения осуществляется по ее имени. Если хранимая процедура является частью группы, то вызов ее осуществляют с помощью команды EXEC. Например,

```
EXEC proc; 3
```

Пример 1. Создание и использование хранимой процедуры.

Создадим процедуру, выполняющую отбор записей из таблицы Employee, в которой условия отбора определяются с помощью двух параметров.

```
CREATE PROCEDURE procSelect
(@p1 char(30), @p2 char (20))
AS
SELECT Name, Department, Cost FROM Employee
WHERE @p1=@p2
```

Обращение к процедуре может иметь следующий вид:

```
procSelect Name, 'Kooper'
```

Возможным результатом такого обращения будет строка вида:

Name	Department	Cost
Kooper	Trade	3000

Создание хранимой процедуры с помощью программы SQL Server Enterprise Manager включает следующие действия.

1. Запуск этой программы из группы программ Microsoft SQL Server.
2. Выбор в открывшемся диалоговом окне программы сервера баз данных и базы данных.
3. Выбор элемента **Stored Procedures** (**Хранимые процедуры**) и выполнение команды его контекстного меню **New Stored Procedure** (**Создать хранимую процедуру**).
4. В открывшемся диалоговом окне **Stored Procedure Properties** (**Свой-**

ства хранимой процедуры) в поле **Text** (Текст) ввод операторов Transact-SQL создаваемой процедуры и указание имени процедуры на месте фразы <PROCEDURE NAME> (рис. 13.3).

5. Нажатие кнопки **Check Syntax** (Проверка синтаксиса) для проверки отсутствия синтаксических ошибок и при необходимости корректировка операторов для устранения ошибок.
6. Нажатие **OK**, в результате чего хранимая процедура создается и сохраняется.

Триггеры представляют собой разновидность хранимых процедур, запуск

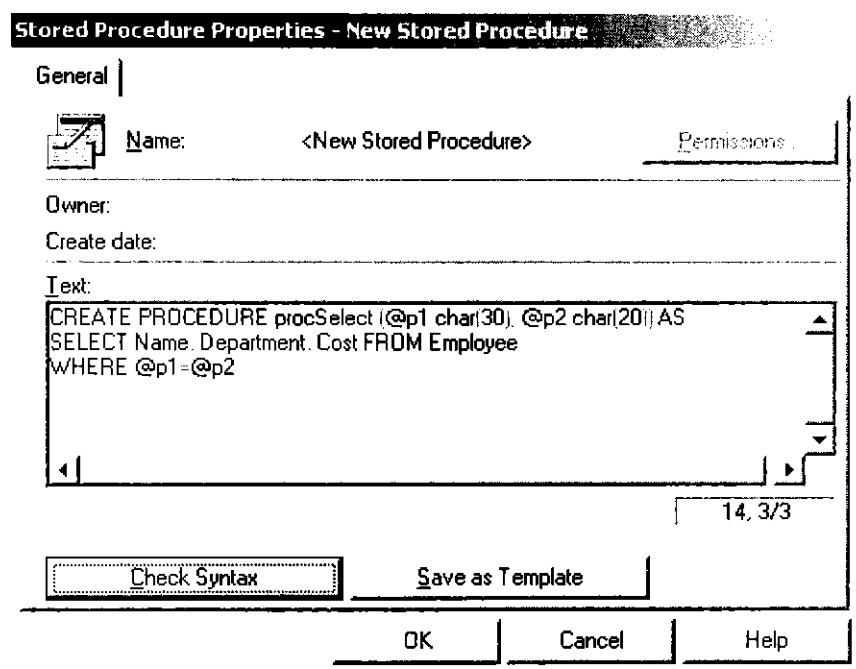


Рис. 13.3. Окно создания хранимой процедуры

емых автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Триггер запускается автоматически при вставке, модификации или удалении данных в определенной таблице.

Наиболее часто триггеры применяются для сложных критериев в базе данных в случае, когда недостаточно стандартных ограничений и табличных средств по обеспечению целостности данных. Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т. п. В SQL Server 2000 появилась возможность создавать триггеры и для представлений.

Триггер и вызвавший его оператор Transact-SQL рассматриваются как

единая транзакция, отменяемая (откатываемая) из триггера. Обычно в состав триггера входит набор команд выполнения некоторых действий над данными или проверки определенных условий. При невозможности обработки данных или невыполнимости условий происходит откат транзакции.

Создание триггера возможно владельцем базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т. п.

Для создания триггеров используется оператор **CREATE**, упрощенный формат которого имеет следующий вид:

```
CREATE TRIGGER [владелец.]имя_триггера  
ON [владелец.]имя_таблицы | имя_представления}  
[WITH ENCRYPTION]  
FOR [{AFTER | INSTEAD OF}]  
{INSERT, UPDATE, DELETE}  
AS  
<операторы_SQL>
```

Параметр **WITH ENCRYPTION** (С шифрованием) служит для предотвращения возможности прочтения текста триггера после помещения его на сервер.

После ключевого слова **FOR** указывается тип триггера: стандартный (**AFTER**), запускаемый после выполнения пользователем изменений данных, либо выполняемый взамен команды, приведшей к запуску триггера (**INSTEAD OF**). По умолчанию считается заданным тип AFTER.

Ключевые слова **INSERT** (Вставить), **UPDATE** (Обновить) и **DELETE** (Удалить) определяют операции, которые инициируют выполнение триггера.

SQL Server сохраняет текст триггера в таблице системного каталога **syscomments**.

Пример 2. Создание триггера вставки.

Рассмотрим создание триггера, который выполняется при вставке записи в таблицу **schet**. В таблице **schet** хранится информация о покупках и есть поле, где указывается номер накладной о покупке одной или нескольких единиц товара. Общая сумма стоимости по накладной хранится в отдельной таблице **prod**. Назначением создаваемого нами триггера является срабатывание на добавление записи в таблицу **schet** и выполнение добавления записи в таблицу **prod**, причем номер накладной в обеих таблицах должен совпадать.

```
CREATE TRIGGER schet_insert
```

```
ON schet
FOR INSERT
AS
INSERT INTO prod (schet, sum)
VALUES (vstavka.schet, vstavka.sum)
```

Замечание.

Для обеспечения высокой надежности работы с БД и возможности восстановления баз данных и приложений целесообразно иметь резервные копии хранимых процедур, триггеров, определений таблиц и общей структуры серверной части приложений SQL Server.

Триггеры представляют собой весьма полезное и в то же время опасное средство. Так, при неправильной логике работы триггера можно легко уничтожить целую базу данных. Поэтому требуется очень тщательно отлаживать триггеры и проверять логику их работы.

13.8. Обеспечение безопасности

Систему безопасности SQL Server условно можно разделить на следующие два уровня: сервера и базы данных. На уровне *сервера* определяется возможность доступа пользователей к серверу. На уровне *базы данных* для пользователей, получивших доступ к серверу, устанавливаются права доступа к объектам базы данных.

В SQL Server на уровне сервера используются следующие средства обеспечения безопасности:

- идентификация (*identification*) по имени пользователя при входе в систему;
- аутентификация (*authentication*) – проверка подлинности пользователя с помощью пароля;
- учетная запись (*login*);
- встроенные или фиксированные роли сервера (*fixed server roles*).

На сервере система защиты SQL Server может быть реализована в двух следующих режимах:

- *стандартном* (*Mixed Mode*, режим смешанной аутентификации) – комбинацией средств защиты SQL Server и Windows NT/2k;
- *интегрированном* (*Windows Authentication Mode*, режим аутентификации Windows) – использованием только средств защиты Windows NT/2k.

Названные режимы определяют, как пользователи SQL Server регистрируются на сервере и как они входят в операционную систему Windows NT/2k.

Настройка сервера баз данных на определенный режим функционирования

ния системы защиты выполняется с помощью программы SQL Server Enterprise Manager следующим образом:

1. Запуск программы SQL Server Enterprise Manager.
2. Выбор нужного сервера баз данных, открытие окна его свойств, например, с помощью команды **Properties (Свойства)** контекстного меню и выбор вкладки **Security**.
3. С помощью переключателей группы **Authentication: (аутентификация:)** установка требуемого способа защиты: стандартный – SQL Server and Windows NT/2000, интегрированный – Windows NT/2000 only.
4. С помощью переключателей группы **Audit level: (Уровень аудита:)** выбирается требуемый вариант учета попыток регистрации пользователей:
 - **None** – попытки доступа не протоколируются;
 - **Success** – протоколируются только успешные регистрации;
 - **Failure** – протоколируются неуспешные попытки регистрации;
 - **All** – протоколируются все регистрации.

В зависимости от конфигурации сервера информация системы аудита содержится в журнале приложений операционной системы (Windows NT/2k application log) или в журнале ошибок SQL Server 2000 или в обоих журналах. Просмотреть содержимое журналов можно с помощью утилит Windows и SQL Server Enterprise Manager соответственно.

В *стандартном* режиме защиты контроль и управление учетными записями, используемыми для доступа к серверу, осуществляется SQL Server. Кроме того, SQL Server самостоятельно выполняет аутентификацию пользователей, хранит данные о правах доступа, именах и паролях. Стандартный режим защиты используется наиболее часто. Его рекомендуется применять в случаях, когда в сети не используются средства Windows NT/2k для аутентификации пользователей и при использовании подключения к серверу с помощью различных протоколов.

В *интегрированном* режиме защиты контроль над устанавливаемыми пользователями соединениями осуществляет операционная система Windows NT/2k. При этом используются списки контроля доступа ACL (Access Control List). *Достоинствами* интегрированного режима защиты является то, что после регистрации пользователя в домене (т. е. ввода своего имени и пароля) он сразу получает соответствующие права доступа ко всем ресурсам домена Windows NT/2k, в том числе и к данным SQL Server 2000, а также использование шифрования при передаче по сети. Такой метод автоматического предоставления доступа называется *установлением доверительного соединения*. Считается, что этот режим является более защищенным по сравнению с предыдущим, так как аутентификация средствами Windows NT является гораздо более защищенной, чем аутентификация SQL Server.

Рассмотрим более подробно осуществляющую средствами SQL Server орга-

низацию защиты в *стандартном* режиме. SQL Server в работе использует два уровня доступа пользователей: первый уровень — учетные записи, второй уровень — записи пользователей.

Учетные записи служат для подключения к серверу системы SQL Server, область их действия распространяется на весь сервер. Учетные записи хранятся в таблице sysxlogins системной базы данных master. Учетная запись в SQL Server ассоциируется с паролем, позволяющим получить доступ к любой базе данных сервера.

Записи пользователей служат для контроля над правами доступа к определенным ресурсам сервера, таким как таблицы и хранимые процедуры. Записи пользователя создаются в одной или нескольких базах данных независимо друг от друга. Записи пользователей размещаются в таблице sysusers, имеющейся в каждой базе данных.

В записи пользователя может быть определена роль пользователя — одна или несколько. Понятие роли было введено в SQL Server 7.0 вместо понятия группы SQL Server 6.x. С ролью пользователя связывают его функции защиты, например, пользователю (одному или нескольким) может быть назначена роль администратора системы безопасности, что дает ему право управлять всеми учетными записями в системе. В SQL Server 2000 можно работать как с группами, так и с ролями.

Набор ролей сервера неизменяем. Никто, даже администратор сервера, не может создать новую роль или удалить существующую роль сервера.

Перечислим фиксированные роли сервера:

- **sysadmin** (члены этой роли имеют наиболее широкие права в SQL Server 2000);
- **setupadmin** (члены этой роли могут управлять связанными серверами, конфигурировать запускаемые автоматически при старте сервера хранимые процедуры);
- **serveradmin** (членам этой роли разрешено останавливать сервер (**SHUTDOWN**), изменять параметры работы службы (**sp_configure**), изменять конфигурацию (**RECONFIGURE**) и управлять полнотекстовым поиском (**sp_fulltext_service**));
- **securityadmin** (члены этой роли могут создавать новые учетные записи с правами создания баз данных и их объектов, управлять связанными серверами, включать учетные записи в роль **securityadmin** и читать журнал ошибок SQL Server);
- **processadmin** (членам данной роли дано право управлять процессами SQL Server 2000, в частности выдавать команду **KILL**, и включать другие учетные записи в эту роль);
- **diskadmin** (данная роль используется для управления устройствами предыдущих версий SQL Server);
- **dbcreator** (члены этой роли могут создавать, удалять и переименовывать базы данных).

- вать БД, а также восстанавливать их резервные копии БД и журналов транзакций);
- **bulkadmin** (члены этой роли могут вставлять данные с помощью средств массивной закачки, не имея непосредственного доступа к таблицам).

Создание учетной записи рекомендуется выполнять средствами SQL Server Enterprise Manager либо используя системные хранимые процедуры.

С помощью программы SQL Server Enterprise Manager это может быть сделано следующими действиями:

1. Запуск программы.
2. Открытие в окне программы требуемого сервера, затем его папки **Security** (Безопасность) и выделение объекта **Logins** (Учетные записи).
3. Выполнение команды **Action | New Login** (Действие | Новая учетная запись):
4. В открывшемся диалоговом окне ввод данных о новой учетной записи пользователя (на вкладке **General** (Общие) окна указание имени учетной записи и пароля).
5. На вкладке **Database Access** (Доступ к базе данных) указание баз данных, к которым пользователю разрешается доступ, нажатие **OK**.

Создание записи пользователя в базе данных можно выполнить с помощью SQL Server Enterprise Manager следующими действиями:

1. Запуск программы и выделение в окне программы требуемой базы данных.
2. Выполнение команды **Action | New | Database User** (Действия | Создать | Пользователь базы данных). В результате открывается диалоговое окно **Database User Properties** (Свойства пользователя базы данных).
3. Выбор в списке **Login name:** (Учетная запись:) учетной записи, для которой в базе данных создается запись пользователя.
4. В поле **User name:** (Имя пользователя:) указание имени пользователя.
5. Выбор необходимых для данного пользователя ролей в списке **Database role membership:** (Роли базы данных:).

В базе данных роли могут иметь один из двух типов: **standard** и **application**. Роль типа **standard** назначается пользователю и обеспечивает ему определенные права доступа к объектам базы данных.

Роль типа **application** назначается приложениям для предоставления прав доступа к объектам базы данных. Роли типа **application** требуют указания пароля для получения доступа.

Для управления ролями уровня базы данных можно использовать средства Transact-SQL и интерфейс SQL Server Enterprise Manager. Если пользоваться SQL Server Enterprise Manager, то список существующих ролей БД

можно увидеть в папке Roles произвольной базы. Следует обратить внимание на то, что каждая база данных уже имеет фиксированные роли (fixed database role).

Фиксированные роли базы данных предоставляют пользователям определенные наборы прав, которые нельзя организовать никаким другим способом. Количество и назначение этих ролей стандартно и не может быть изменено. Перечислим фиксированные роли базы данных:

- db_securityadmin (члены этой роли могут управлять правами доступа к объектам БД других пользователей и членством их в ролях);
- db_owner (члены этой роли имеют права владельца, разрешающие любые действия);
- db_ddladmin (члены данной роли могут создавать, изменять и удалять объекты БД);
- db_accessadmin (члены этой роли могут создавать, удалять или изменять пользователей БД);
- db_backupoperator (членам этой роли можно выполнять резервное копирование БД);
- db_datawriter (члены этой роли могут изменять данные в любой таблице или представлении БД);
- db_datareader (членам этой роли разрешено читать данные из любой таблицы или представления БД);
- db_denydatawriter (членам этой роли запрещается изменять данные независимо от предоставленных им разрешений);
- db_denydatareader (членам этой роли запрещается просматривать данные независимо от предоставленных им разрешений).

Право доступа (permission) представляет собой разрешение на получение доступа к определенному объекту базы данных, например, таблице или представлению. Права доступа предоставляются некоторой записи пользователя или роли, разрешая им выполнять определенные операции с объектом базы данных. Для каждого типа объекта базы данных имеется несколько видов прав доступа.

Все права доступа автоматически предоставляются владельцу или создателю объекта базы данных. В дальнейшем владелец может предоставить права доступа другим пользователям или группам.

Наибольшими правами доступа обладают следующие категории пользователей:

- системный администратор (sa) имеет все права доступа ко всем объектам во всех базах данных сервера. Учетная запись sa оставлена в SQL Server 2000 для сохранения обратной совместимости с предыдущими версиями;
- члены группы администраторов домена, в котором установлен локаль-

ный SQL Server 2000. Учетная запись этой группы имеет имя BUILTIN\Administrators. По умолчанию она включается в фиксированную роль сервера, что означает предоставление по умолчанию администраторам сети прав управления сервером;

- владелец базы данных (Database Owner) имеет все права доступа ко всем объектам его базы данных.

Право доступа к объекту (object permission) представляет собой разрешение на выполнение конкретных действий над объектами базы данных, например, таблицами, представлениями и хранимыми процедурами. В частности, для таблиц существуют следующие типы прав доступа: SELECT – разрешение выборки данных; INSERT – разрешение вставлять данные; UPDATE – разрешение изменять данные и т. д.

Задание и отмену прав доступа к объектам базы данных можно выполнить с помощью команды GRANT и REVOKE языка Transact SQL и с помощью программы SQL Server Enterprise Manager.

Изменение прав доступа к объектам базы данных с помощью программы SQL Server Enterprise Manager включает следующие действия:

1. Запуск программы и выделение в окне программы требуемой базы данных, для которой выполняется изменение прав доступа к объектам.
2. Выполнение команды Action | Properties (Действия | Свойства). В результате откроется окно свойств базы данных.
3. На вкладке Permission (Разрешения) выбор требуемого пользователя или роли и установление требуемых прав доступа к объектам с помощью флажков, нажатие OK.

13.9. Организация взаимодействия клиент-сервер

Напомним, что при использовании технологии клиент-сервер приложение разделяется на две (или более) части. Клиентская часть (Front-end) обеспечивает удобный графический интерфейс и размещается на компьютере пользователя. Серверная часть (Back-end) осуществляет управление данными, разделение информации, администрирование и обеспечивает безопасность информации. Клиентское приложение формирует запросы к серверу базы данных, на котором выполняются соответствующие команды. Результаты выполнения запроса пересылаются клиенту.

При разработке распределенных информационных систем в организации взаимодействия клиентской и серверной части можно выделить следующие важные в практическом смысле задачи:

- *перенос персональной базы данных* на сервер для последующего ее коллективного использования как корпоративной базы данных;
- *организация запросов* к корпоративной базе данных, размещенной на сервере, со стороны компьютера-клиента;
- *разработка клиентского приложения* для удаленного доступа к корпоративной базе данных со стороны компьютера-клиента;
- *администрирование* сервера со стороны клиента.

Рассмотрим названные задачи более подробно.

Перенос персональной базы данных на сервер

Задача переноса персональной базы данных на сервер может возникать в ситуациях, когда требуется обеспечить коллективный доступ к базе данных, разработанной с помощью персональной СУБД, такой как Microsoft Access или Microsoft Visual FoxPro. Для решения этой задачи в составе названных персональных СУБД имеются соответствующие средства. Так, в Microsoft Access 2002 имеется Upsizing Wizard (Мастер «наращивания»), предназначенный для преобразования базы данных Access к формату SQL Server. В составе пакета Microsoft Visual FoxPro 8.0 имеется аналогичный Мастер, предназначенный для преобразования базы данных Visual FoxPro к формату SQL Server или Oracle.

Способы взаимодействия при подготовке запросов к базе данных

Подготовка запросов к базе данных на сервере (на языке SQL) со стороны клиентской части может выполняться с помощью некоторой утилиты, например, Query Analyzer. Для предоставления пользователю больших возможностей и удобства в подготовке и выполнении запросов создаются клиентские приложения.

Для *организации запросов к серверной базе данных* непосредственно на языке SQL или с помощью клиентского приложения возможны различные способы взаимодействия, заметно влияющие на его эффективность. К числу основных способов такого взаимодействия можно отнести следующие способы, основанные на использовании:

- интерфейса DB-Library или DB-LIB (библиотек баз данных);
- технологии ODBC (совместимости открытых баз данных);
- интерфейса OLE DB (связывания и встраивания объектов баз данных);
- технологии DAO (Data Access Object – объектов доступа к данным);
- технологии ADO (ActiveX Data Object – объектов данных ActiveX).

Названные интерфейсы и технологии могут применяться совместно, например, ODBC и DAO или ODBC, ADO и OLE DB. Охарактеризуем подробнее названные подходы к организации взаимодействия клиентской и серверной частей баз данных.

Интерфейс DB-Library представляет собой специализированную библиотеку функций API, осуществляющих прямой доступ к SQL Server из среды систем программирования C и Visual Basic. Они предоставляют средства отправки запросов и получения информации от SQL Server.

Работа с библиотекой DB-LIB предполагает следующую стандартную последовательность действий: регистрацию в системе (установление соединения), выполнение нескольких действия на сервере и отключение сервера (закрытие соединения).

DB-LIB представляет собой специально предназначенный для SQL Server интерфейс прикладных программ. Поэтому он является наименее мобильным из числа рассматриваемых в смысле возможностей переноса приложений в другую серверную среду. С точки зрения производительности этот способ позволяет осуществить самый быстрый доступ к информации. Причиной этого является то, что он предоставляет оптимизированный интерфейс прикладного программирования и непосредственно использует язык запросов системы SQL Server.

DB-LIB представляет интерфейс SQL Server уровня прикладного программирования. Отметим, что этот уровень не используется непосредственно в среде персональных СУБД, таких как Access и Visual FoxPro. В них используются абстрактные промежуточные уровни ODBC или OLE DB, упрощающие доступ к базам данных.

Технология DB-LIB использована в большом числе разработанных ранее информационных систем. В настоящее время эта технология не поддерживается фирмой Microsoft и вместо нее рекомендуется использовать средства OLE DB.

Технология ODBC, как отмечалось ранее, разработана фирмой Microsoft для обеспечения возможности взаимосвязи между различными СУБД. Она предусматривает создание дополнительного уровня между приложением и используемой СУБД. Службы ODBC обеспечивают получение от приложения запросов на выборку информации, перевод их на язык ядра адресуемой базы данных для доступа к хранимой в ней информации.

Основное назначение ODBC состоит в абстрагировании приложения от особенностей ядра серверной базы данных, с которой оно осуществляет взаимодействие, поэтому серверная база данных становится как бы прозрачной для любого клиентского приложения.

Достоинством технологии ODBC является простота разработки приложений, обусловленная высоким уровнем абстрактности интерфейса доступа к данным практически любых существующих типов СУБД (см. подр. 4.4). При этом возможно создание источника данных, связанного с любым типом базы данных. Используя эту технологию, можно создавать клиент-серверные приложения, причем средствами персональных СУБД целесообразно разрабатывать клиентскую часть приложения, а средствами SQL Server — серверную часть.

Основной недостаток технологии ODBC связан с необходимостью трансляции запросов, что снижает скорость доступа к данным. В системах клиент-сервер этот недостаток устраняется путем перемещения обработки запроса с компьютера-клиента на компьютер-сервер. При этом устраняются промежуточные звенья, являющиеся основной причиной снижения скорости обработки информации с использованием рассматриваемой технологии.

OLE DB представляет собой интерфейс прикладного программирования API, который позволяет приложениям COM (Component Object Model – объектная модель компонентов) потреблять данные из источника данных OLE DB. Источник данных OLE DB включает данные, хранимые в различных форматах, не только в формате баз данных SQL. Приложение использует *провайдер OLE DB* для доступа к источнику данных OLE DB. Провайдер OLE DB представляет собой компонент COM, позволяющий принимать вызовы OLE DB API и выполнять все необходимое для обработки запроса к источнику данных.

В ранних версиях SQL Server приложения OLE DB должны были использовать провайдер OLE DB для ODBC, основанного на драйвере Microsoft SQL Server ODBC. Поскольку приложения по-прежнему могут использовать провайдер OLE DB для ODBC, поэтому более эффективно использование только провайдера OLE DB для SQL Server.

Интерфейс прикладного программирования OLE DB рекомендуется использовать для создания средств и утилит, или разработок системного уровня, нуждающихся в высокой производительности или доступе к свойствам SQL Server, недоступным с помощью технологии ADO. Основные возможности спецификации OLE DB обеспечивают полную функциональность доступа к данным, требуемую большинством приложений. Кроме того, OLE DB позволяет индивидуальным провайдерам определять специфические механизмы поддержки дополнительных свойств процессора данных, доступного провайдеру. Приложения ADO могут не иметь доступа к некоторым свойствам SQL Server, проявляемым через специфические свойства провайдера OLE DB для SQL Server. Поэтому приложения, желающие воспользоваться такими специфическими свойствами, должны использовать OLE DB API.

В SQL Server 2000 процессор баз данных сервера использует OLE DB для связи: между внутренними компонентами, такими как процессор хранения и процессор отношений; между установками SQL Server при использовании удаленных хранимых процедур; как интерфейс к другим источникам данных для распределенных запросов.

Технология DAO означает использование объектов, методов и свойств, которые существенно упрощают работу приложения с базой данных. Для обмена информацией с SQL Server применяются уровни доступа процессора баз данных Jet SQL Server и ODBC. Кроме того, при этом создается еще один уровень абстракции между приложением и функциями ODBC, используемыми при выполнении запросов.

При использовании технологии DAO работа с базами данных, таблицами, представлениями и т. д. ведется с использованием коллекций объектов. При этом обеспечиваются большие удобства в работе с объектами баз данных. К примеру, для создания представления проще вызвать метод Add соответствующего представлению объекта, чем применять стандартные средства ODBC с указанием имен используемых для этих целей хранимых процедур.

При работе с базами данных технология DAO применима для большинства источников данных, поддерживаемых средствами ODBC. Его можно применять также на других платформах и в системах программирования. В частности, технология DAO поддерживается версиями Visual Basic.

В настоящее время технология DAO постепенно вытесняется технологией ADO, которая позволяет разрабатывать приложения Web для работы с базами данных.

Технология ADO основана на объектной модели, в которой объекты имеют наборы коллекций, методов и свойств, обеспечивающие поддержку баз данных. Объекты этой технологии предоставляют наиболее широкие возможности по интеграции приложений с базами данных. С использованием технологии ADO приложения для работы с базами данных можно создавать в различных системах программирования, например, Visual Basic и Visual Basic for Application. Кроме того, объекты можно использовать при разработке Web-приложений для среды ASP (Active Server Pages — активных серверных страниц), или приложений ASP.

Объекты ADO доступны в среде ASP и функционируют на уровне OLE DB. При этом технология ADO обеспечивает установление соединений ODBC и работу с уровнем OLE DB. При организации доступа к базе данных из приложения ASP инициатором установки соединения с сервером базы данных является клиентское приложение. Причем, приложения ASP выполняются в среде Internet Information Server, а не на SQL Server.

В целом технологию ADO можно характеризовать как наиболее современную технологию разработки приложений для работы с распределенными базами по технологии клиент-сервер.

Администрирование сервера со стороны клиента

Для администрирования сервера со стороны клиента могут использоваться компоненты клиентской части MS SQL Server 2000, например, программа SQL Server Enterprise Manager. Кроме того, для этих целей может использоваться так называемая технология SQL-DMO.

Технология SQL-DMO (SQL Distributed Management Objects — объекты распределенного управления SQL) позволяет приложениям, написанным на языках программирования, поддерживающих OLE или COM, выполнять администрирование всеми компонентами SQL Server. SQL-DMO представля-

ет собой интерфейс прикладного программирования API, используемый SQL Server Enterprise Manager. Поэтому приложения, используя SQL-DMO, могут выполнять все функции, выполняемые программой SQL Server Enterprise Manager.

При использовании технологии SQL-DMO работа ведется с объектами, коллекциями, свойствами и методами объекта SQL-DMO. Кроме того, возможно выполнение и посылка на сервер команд SQL Server с помощью вызова методов `ExecuteImmediate` и `ExecuteWithResults`.

13.10. Обработка данных с помощью ODBC

Рассмотрим организацию обработки в базе данных на сервере SQL Server с помощью технологии ODBC и использовании Access в качестве клиентского приложения. При использовании в качестве клиентского приложения других персональных СУБД, например, Visual FoxPro общая технология организации обработки данных сохраняется.

При использовании в клиентском приложении средств ODBC осуществляется обращение к определенному источнику данных, а через него — к СУБД, которую он представляет. При установке средств ODBC устанавливается общая подсистема ODBC и определяются пары «драйвер – база данных», которым задаются имена, используемые при установке соединения с базой данных. Соответствующие пары называются *DSN* (Data Source Names — имена источников данных, или *поименованные источники данных*).

Каждый источник данных описывает собственно источник данных и информацию о доступе к этим данным. В качестве данных могут выступать базы данных Microsoft Access, Microsoft SQL Server, Oracle RDBMS, электронные таблицы, а также обычные текстовые файлы. Информация о доступе, например, к серверу БД, обычно включает в себя сведения о размещении сервера, имя БД, идентификатор учетной записи и пароль, а также различные параметры драйвера, описывающие как устанавливать соединение с источником данных.

При обработке данных на сервере с использованием технологии ODBC и применением клиентского приложения можно выделить следующие два основных этапа: задание источника данных — создание и настройка соединения, а также собственно обработка данных с помощью запросов.

Создание и настройка источника данных

Создание источника данных выполняется с помощью утилиты ODBC Data Source Administrator, вызываемой из окна панели управления (Control Panel). С ее помощью (рис. 13.4) можно создать новое соединение и настроить параметры имеющихся соединений.

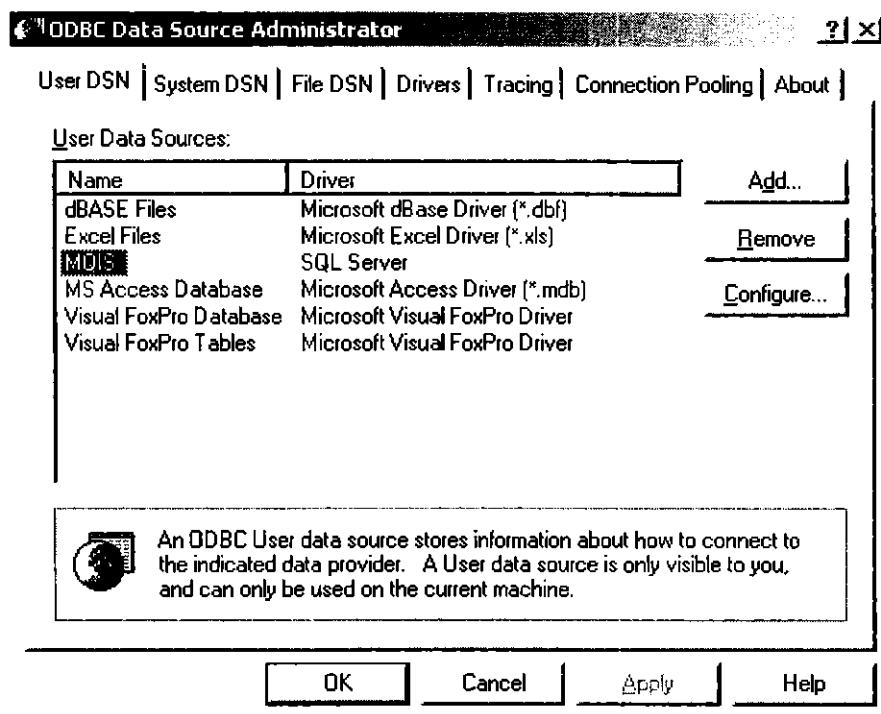


Рис. 13.4. Окно настройки параметров соединения

Для создания нового соединения нужно в окне утилиты ODBC Data Source Administrator (рис.13.4) нажатием кнопки Add... (Добавить...) вызвать Wizard (Мастер) создания источника данных ODBC и в его диалоговых окнах указать параметры. В их число входят: имя и описание источника данных; сервер, с которым устанавливается соединение; метод аутентификации; имя базы данных; имя базы данных по умолчанию и некоторые другие. Желательно указать имя используемой базы данных. Это гарантирует то, что при установке соединения будет выбрана именно эта база данных.

Для изменения параметров имеющегося соединения достаточно в окне утилиты ODBC Data Source Administrator (рис. 13.4) выбрать нужное соединение, нажатием кнопки Configure... (Настройка...) вызвать диалог настройки параметров соединения и внести нужные изменения.

Источники данных ODBC могут быть следующих трех видов (все поддерживаются СУБД MS Access):

- пользовательские (вкладка User DSN окна утилиты), специфичные для каждого пользователя компьютера-клиента. Их можно создавать с помощью утилиты ODBC Administrator и с помощью Access. Такие источ-

- ники данных могут использоваться только тем пользователем некоторого компьютера, которым они создавались;
- системные (вкладка **System DSN** окна утилиты), — также можно создавать с помощью утилиты ODBC Administrator и с помощью Access. Такие источники данных могут быть доступны многим пользователям (с соответствующими привилегиями) некоторого компьютера, а также службам Windows NT/2k;
 - файловые (вкладка **File DSN** окна утилиты) — являются текстовыми файлами, определяющими источник данных. Такие источники данных могут использоваться в режиме совместного доступа многими пользователями, имеющими аналогичные установленные драйверы.

Использование баз данных

После задания базы данных SQL Server в качестве источника данных (пользовательского, системного или файлового) можно выполнить подключение, импорт или экспорт таблиц базы данных SQL Server в базу данных Access. При этом учитываются права доступа, предоставляемые SQL Server пользователям. При использовании таблиц серверной базы данных не требуется указывать используемые индексы, так они считаются автоматически при открытии таблицы.

Подключение таблиц базы данных SQL Server, например dbs, включает следующую последовательность действий:

1. Запуск Access и создание базы данных с произвольным именем, например, dba.mdb.
2. С помощью команды меню **Файл | Внешние данные | Связь** (File | Get External Data | Link Tables) открытие диалогового окна Связь (Link).
3. В списке Тип файла (Files of Type) выбор элемента Базы данных ODBC (ODBC Databases).
4. В очередном диалоговом окне **Select Data Source** (Выбор источника данных) на вкладке Machine Data Source (Машинный источник данных) выбор двойным щелчком источника данных в списке.
5. В очередном диалоговом окне **SQL Server Login** (Регистрация SQL Server) указание имени и пароля, нажатие OK для регистрации в SQL Server. После подключения к базе данных dbs с помощью ODBC API открывается диалоговое окно Link Tables (Связь с таблицами), в котором приводится список имен таблиц с префиксами dbo (database owner — владелец базы данных).
6. Подключение при необходимости всех таблиц нажатием кнопки Select All (Выделить все), затем OK.
7. Для обновления таблиц Access нужно иметь уникальный индекс. Если в таблице SQL Server нет индекса первичного ключа, то при попытке

обновления таблицы открывается диалоговое окно **Select Unique Record Identifier** (Выбор уникального индекса). В нем задаются поля первичного ключа, и нажимается **OK**.

Подключенные таблицы выводятся в списке таблиц в окне базы данных. Знаком глобуса слева от имени таблицы указывает на подключение с помощью ODBC. В именах таблиц вместо точек используется символ подчеркивания.

Создание запроса к базе данных SQL Server включает следующие шаги:

1. Создание нового запроса и добавление в него требуемых таблиц из базы данных SQL Server, например, `dbs.mdb`.
2. Перетаскивание требуемых полей из добавленных таблиц в строку **Field** (Поле) бланка запроса.
3. Нажатие на панели инструментов кнопки **Run** (Запуск) для вывода результатов выполнения запроса.
4. Закрытие и сохранение запроса.

Приведенная выше процедура подключения таблиц используется в большинстве случаев для баз данных SQL Server при наличии драйверов ODBC.

В SQL Server можно писать хранимые процедуры и выполнять их вместо посылки серверу отдельных команд. Выполнение запроса в виде хранимой процедуры осуществляется быстрее, чем с помощью ядра базы данных Jet. Это обусловлено уменьшением объема передаваемых данных и тем, что хранимые процедуры выполняются быстрее, поскольку предварительно компилируются. Кроме того, запрос SQL к серверу используется при необходимости использовать средства Transact-SQL, отсутствующие в SQL Access.

Для преобразования запроса Access в запрос SQL к серверу выполняется следующее:

1. Открытие ранее созданного запроса в режиме Конструктора запросов.
2. Выполнение команды **Query | SQL-Specific | Pass-Through** (Запрос | Запрос к серверу | К серверу). В результате открывается диалоговое окно, содержащее команду **SQL**.
3. Из каждого имени таблицы в тексте команды запроса SQL удаление префикса `dbo_`.
4. Сохранение запроса к серверу с помощью команды меню **File | Save** (Файл | Сохранить).
5. Нажатие на панели инструментов кнопки **Run** (Запуск) для выполнения запроса к серверу. В открывшемся диалоговом окне **Select Data Source** (Выбор источника данных) выбор нужного источника данных, нажатие **OK**.
6. Задание в очередном окне имени и пароля, нажатие **OK**. Результатирующее множество запроса будет таким же, как и при выполнении стандартного запроса ядром SQL Server.

Контрольные вопросы и задания

1. Перечислите основные нововведения в MS SQL Server 2000 и поясните в чем их суть.
2. Назовите службы MS SQL Server 2000 и опишите их функции.
3. Каковы режимы работы MS SQL Server 2000?
4. Какие имеются инструменты в составе MS SQL Server 2000 и для чего они предназначены?
5. Назовите варианты поставки MS SQL Server 2000 и требования, предъявляемые к аппаратному и программному обеспечению.
6. Охарактеризуйте понятия домена и рабочей группы.
7. Дайте общую характеристику языку запросов Transact-SQL.
8. Поясните способы задания операторов Transact-SQL.
9. Укажите назначение и дайте краткую характеристику системным базам данных MS SQL Server 2000.
10. Для чего предназначен каталог баз данных и системный каталог?
11. Каким образом может быть выполнено создание базы данных пользователем?
12. Назовите способы создания таблиц в MS SQL Server 2000.
13. В чем разница в создании и использовании локальной и глобальной временных таблиц?
14. Охарактеризуйте типы данных MS SQL Server 2000.
15. Поясните способы создания таблиц.
16. Назовите правила неявного присваивания значения полю.
17. Как осуществляется добавление данных в таблицы и выборка данных из таблиц?
18. Охарактеризуйте типы индексов таблиц MS SQL Server 2000.
19. Какие рекомендации нужно учитывать при выборе столбца для создания индекса?
20. Опишите создание индекса с помощью программы SQL Server Enterprise Manager.
21. Каким образом выполняется добавление и удаление ключей?
22. Охарактеризуйте хранимые процедуры; поясните, в чем состоит достоинство их применения.
23. Назовите и охарактеризуйте типы хранимых процедур, различаемые по области видимости.
24. Как осуществляется создание, использование и удаление хранимых процедур?
25. Что представляют собой триггеры, как они создаются?
26. Как соотносятся стандартный и интегрированный режимы защиты?
27. Дайте характеристику понятий учетной записи, записи пользователя и роли.

28. Охарактеризуйте права доступа к объектам и опишите технологию их задания.
29. Охарактеризуйте способы организации взаимодействия клиент-сервер при подготовке запросов к базе данных на сервере.
30. Опишите создание источника данных при использовании технологии ODBC.
31. Как осуществляется подключение баз данных с помощью ODBC?
32. Выполнить разработку БД и приложения для работы по технологии клиент-сервер:
 - создать БД в среде персональной СУБД Access или Visual FoxPro;
 - выполнить перенос персональной БД на MS SQL Server с помощью Upsizing Wizard (Мастер «наращивания»);
 - разработать запросы к данным сервера со стороны клиентской части (СУБД Access или Visual FoxPro), используя интерфейс ODBC.

Литература

1. Артемов Д., Погульский Г. Microsoft SQL Server 7.0: установка, управление, оптимизация. -- М.: Издательский отдел «Русская редакция» ТОО @Channel Trading Ltd. – 1998. – 488 с.
2. Винкел С. Использование Microsoft SQL Server 7.0 Специальное издание / Пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 1999. – 816 с.
3. Гарсия М. Ф., Рединг Дж., Уолен Э., Делюк С. А. Microsoft SQL Server 2000. Справочник администратора. – М.: СП ЭКОМ, 2004.
4. Горев А., Макашарипов С., Владимиров Ю. Microsoft SQL Server 6.5 для профессионалов. – СПб.: Питер, 1998. – 446 с.
5. Дженингс Р. Microsoft Access 97 в подлиннике. Том II / Пер. с англ. -- СПб.: БХВ-Санкт-Петербург, 1997. – 688 с.
6. Дибетта П. Знакомство с Microsoft SQL Server 2005. – М.: Русская Редакция, 2005.
7. Мамаев Е.В. Администрирование MS SQL Server 7.0. – СПб.: БХВ-Санкт-Петербург, 2000. – 496 с.
8. Мамаев Е.В. Microsoft SQL Server 2000. – СПб.: БХВ-Санкт-Петербург, 2001. – 1280 с.

ПУБЛИКАЦИЯ БАЗ ДАННЫХ В ИНТЕРНЕТЕ

14. Введение в технологии публикации

Дальнейшее развитие технологий Интернета и СУБД предоставляет дополнительные возможности по использованию информационных ресурсов сети Интернет. Одним из способов электронного представления данных во всемирной сети является публикация баз данных в Интернете, позволяющая размещать информацию из баз данных на Web-страницах сети.

Публикация баз данных в Интернете может потребоваться для решения перечисленных ниже задач, возникающих перед разработчиками программного обеспечения всемирной сети.

- *Организация взаимосвязи СУБД, работающих на различных платформах.* Существует множество информационных систем работающих на различных платформах. Задачи организации их взаимодействия могут возникать при обеспечении различных видов информационного обмена между этими системами.
- *Построение информационных систем в сети Интернет на основе многоуровневой архитектуры БД.* Архитектура таких систем включает дополнительный уровень (Web-сервер с модулями расширения серверной части), реализующий возможность информационного обмена и публикации БД в глобальной сети.
- *Построение локальных интранет-сетей на основе технологии публикации БД в Интернете.* При этом локальные сети строятся на принципах Интернета с наличием при необходимости выхода в глобальную сеть.
- *Использование в Интернете информации из существующих локальных сетевых баз данных.* Эти задачи возникают при необходимости опубликования в глобальной сети информации из локальных сетей Интранет.
- *Применение БД для упорядочивания, каталогизации информации.* Огромный объем информации, представленной в Интернете, не обладает требуемой степенью структурированности, что делает весьма сложным и долгим процесс поиска необходимой информации.
- *Применение языка SQL для поиска необходимой информации в БД.*

- Использование средств СУБД для обеспечения безопасности данных, разграничения доступа и управления транзакциями при создании Интернет-магазинов, защищенных информационных систем и т. д.
- Стандартизация пользовательского интерфейса на основе применения обозревателей Web с типовым внешним видом и типовой реакцией на действия пользователя.
- Использование обозревателя Web в качестве дешевой клиентской программы для доступа к БД.

Размещение информации из БД во всемирной сети стало возможным в связи с развитием технологий Интернета. В следующем разделе рассматриваются важнейшие элементы Интернет-технологий, являющиеся основой для разработки Web-приложений.

14.1. Основы Интернет-технологий

В Интернете вся информация размещается на Web-страницах, написанных на языке HTML (в этом случае имеем HTML-страницы) или его расширениях, таких как DHTML (Dynamic HTML – динамический HTML) и XML (eXtensible Markup Language – расширяемый язык разметки). В содержимое Web-страницы может входить текстовая информация, ссылки на другие Web-страницы, графические изображения, аудио-, видеинформация и другие данные. Эти страницы хранятся на Web-сервере.

Для доступа к Web-страницам используются специальные клиентские программы – обозреватели Web (программы просмотра, или броузеры – от англ. browser), находящиеся на компьютерах пользователей Интернета. Обозреватель формирует запрос на получение требуемой страницы или другого ресурса с помощью специального адреса URL (Universal Resource Locator – универсальный указатель ресурса). Этот адрес определяет тип протокола для передачи этого ресурса, имя домена, используемое для доступа к требуемому Web-узлу, номер порта (порт – логический канал связи, номера определяются стандартами Интернета), локальный путь к файлу и дополнительные аргументы.

В функции Web-обозревателя входит отображение Web-страниц, которые формирует Web-сервер. При этом Web-обозреватель устанавливает соединение с требуемым Web-узлом, используя протокол передачи данных HTTP.

Для расширения возможностей клиентской части (обозревателя) и серверной части создаются программы расширения обозревателя и сервера. Схема взаимодействия обозревателя и сервера с использованием программ расширения приведена на рис. 14.1.

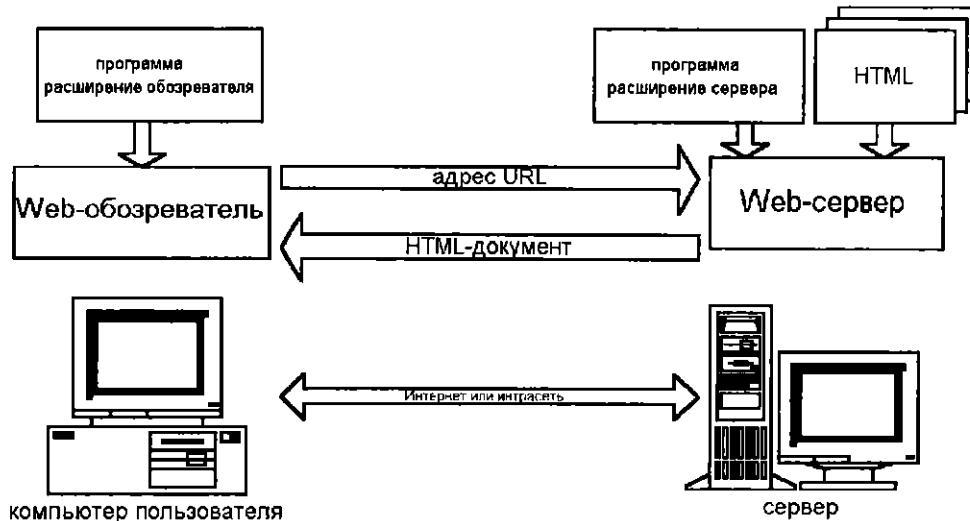


Рис. 14.1. Схема взаимодействия обозревателя и сервера

При организации такого взаимодействия могут использоваться следующие средства:

- сценарии, подготовленные на различных языках сценариев (JavaScript, JScript и VBScript) и вставленные в обычный Web-документ;
- аплеты и сервлеты Java;
- элементы управления ActiveX;
- консольные exe-программы, реализованные с использованием интерфейса CGI;
- exe-программы, реализованные с использованием интерфейса WinCGI;
- динамические библиотеки, реализованные с использованием интерфейса ISAPI;
- динамические страницы IDC/HTX;
- активные серверные страницы ASP;
- персональные домашние страницы PHP.

Дадим краткую характеристику каждого из названных средств.

Сценарии JavaScript, JScript и VBScript

Сценарии, написанные на языках JavaScript, JScript или VBScript, используют для динамического управления интерфейсными объектами (компонентами) Web-документа. Эти языки являются языками интерпретируемого типа (код выполняется в процессе интерпретации). Интерпретация и выполнение сценариев осуществляется обозревателем или сервером. Сценарии являются расширением языка HTML и могут включаться в тело Web-документа.

Заданная часть сценария может исполняться во время загрузки Web-документа, а часть сценария, реализованная, как правило, в виде функции, может выполняться в ответ на действия пользователя. Использование того или иного языка сценариев определяется типом применяемого обозревателя.

JavaScript представляет собой объектно-ориентированный язык с С-подобным синтаксисом. Сценарии на языке JavaScript используются в обозревателе Netscape Navigator, а сценарии на языке JScript используются в обозревателе Internet Explorer. Лексика и синтаксис этих языков практически идентичны, отличия заключаются в используемых объектных моделях обозревателей (совокупностях элементов, их атрибутов и событий Web-документа). В обозревателе Internet Explorer дополнительна поддерживается язык сценариев VBScript, по возможностям функционально эквивалентный языку JScript, но более простой в освоении. Этот язык наиболее часто используется программистами, имеющими опыт работы с Microsoft Visual Basic.

Сценарии могут применяться как расширение обозревателя (клиентское расширение) или как расширение сервера. В случае клиентского расширения сценарии находятся в Web-документе и применяются для создания динамических эффектов при просмотре Web-страницы. На стороне сервера сценарии используются при динамическом создании Web-документов в ответ на запрос пользователя. Для уменьшения нагрузки на Web-сервер часть функций, связанных с предварительной обработкой запросов и введенных данных, целесообразно выполнять на стороне клиента (в Web-обозревателе). В этом случае выделяют *активность на стороне клиента*. Клиентская активность может быть осуществлена и другими средствами (апплеты Java, элементы управления ActiveX).

Элементы управления ActiveX

Элементы управления ActiveX представляют вид модулей расширения, который может использоваться на стороне клиента или на стороне сервера. Они реализуются с помощью динамических библиотек DLL и могут быть встроены в Web-документ как дополнительные интерфейсные элементы. Механизм работы элементов управления ActiveX позволяет из программного кода этих объектов получать неограниченный доступ к локальным ресурсам компьютера пользователя. Из элемента управления ActiveX имеется возможность передавать на сервер любую информацию с компьютера пользователя. Поэтому использование этих элементов на стороне клиента не всегда оправдано в сети Интернет с точки зрения обеспечения безопасности данных.

В коде объектов ActiveX имеется потенциальная возможность наличия вируса. Если загрузить такой объект, то компьютеру клиента в принципе может быть причинен вред. Для частичного устранения этого недостатка ком-

пания Microsoft добавляет в свои коды цифровую подпись (signing), которая обозначает, что данный объект создан надежной компанией. При этом обозреватель Internet Explorer, используя такой объект, выдает предупреждающее сообщение с предоставлением возможности пользователю отменить выполнение кода этого объекта. Этот механизм предназначен для идентификации модулей ActiveX и не устраниет возможность заражения вирусом компьютеров пользователей, использующих объекты ActiveX.

Кроме того, при загрузке текущей Web-страницы, если на ней используются новые элементы управления ActiveX, для их активизации требуется сначала их скачать и установить. А это нарушает принцип универсальности обозревателя. По этим причинам более предпочтительно использовать элементы управления ActiveX на стороне сервера для наращивания его возможностей.

Апплеты и сервлеты Java

Апплеты Java применяются для создания динамически формируемого интерфейса пользователя. Язык Java является объектно-ориентированным языком с синтаксисом, похожим на синтаксис языка C++. Однако возможности языка Java по доступу к локальным ресурсам пользователей сильно ограничены, что делает его безопасным для использования в сети. Апплеты предназначены для выполнения на любых платформах. Их код интерпретируется виртуальной Java-машиной, входящей в состав обозревателя. Использование такого механизма гарантирует целостность локальных данных пользователей. Для использования апплета на Web-странице применяется специальный тег, позволяющий вставлять объект-апплет в любое место Web-документа. Сервлеты, в отличие от апплетов, выполняются на стороне сервера и служат для обработки запросов от обозревателя.

Интерфейсы CGI и WinCGI

Для создания модулей расширения Web-сервера могут использоваться *интерфейсы CGI* (Common Gateway Interface – общий шлюзовой интерфейс) или *интерфейсы программирования API* (Application Program Interface – интерфейс прикладного программирования).

Интерфейс CGI является стандартным протоколом взаимодействия между Web-сервером и модулями расширения, которые могут применяться для выполнения дополнительных функций, не поддерживаемых сервером. Например, такие модули используются для обработки получаемой от пользователя информации, для динамического формирования Web-документа, публикации БД на Web-странице и т. д.

Интерфейсу CGI соответствуют обычные консольные приложения операционной системы DOS. Обмен информацией между сервером и модулем расширения осуществляется с помощью стандартного потокового

ввода/вывода, передача управляющих параметров организуется через переменные окружения операционной системы или через параметры URL-адреса модуля расширения. В качестве стандартных устройств ввода и вывода в среде DOS по умолчанию используются клавиатура и терминал (монитор) соответственно. В этом случае вывод на стандартное устройство перенаправляют в буфер, из которого данные с помощью протокола HTTP отправляются Web-обозревателю.

Для запуска модуля расширения достаточно задать его URL-адрес в строке адреса обозревателя и начать загрузку документа. При получении запроса обозревателя к CGI-приложению сервер запускает это приложение и передает ему данные из командной строки запроса. CGI-приложение формирует ответ и помещает его в выходной поток (на стандартном устройстве вывода), затем сервер посыпает этот ответ с использованием протокола HTTP обратно обозревателю. В случае параллельной обработки нескольких запросов сервер запускает отдельный процесс для обработки каждого запроса. Причем для каждого процесса создается копия модуля расширения в памяти компьютера, на котором находится Web-сервер. Поэтому недостатками этого протокола является невысокая скорость обработки запросов и повышенная загрузка Web-сервера.

Существует адаптированный вариант общего *интерфейса* для среды Windows 3.1 – *WinCGI*. Этот интерфейс отличается от интерфейса CGI тем, что управляющие параметры передаются через INI-файл, а входной и выходной потоки данных перенаправлены в специальные файлы. В остальном механизм взаимодействия с сервером аналогичен механизму, используемому интерфейсом CGI.

Замечание.

Используется также интерфейс FastCGI, который близок к интерфейсу CGI, но в то же время обладает преимуществами интерфейсов ISAPI/NSAPI (рассмотрены ниже). Приложения, написанные в соответствии с ним, запускаются как отдельные изолированные процессы, но являются постоянно активными и не завершаются после обращения к данным, как CGI-сценарии.

Интерфейсы ISAPI/NSAPI

Более перспективными являются *интерфейсы ISAPI/NSAPI* (Internet Server API/Netscape Server API), разработанные фирмами Microsoft и Netscape соответственно. Они также предназначены для разработки дополнительных модулей расширения Web-сервера. В случае их использования модули расширения реализуются в виде библиотек DLL. Запуск модуля расширения выполняется сервером в ответ на запрос обозревателя на загрузку URL-адреса этого модуля. Взаимодействие между сервером и модулем расширения осуществляется с помощью специальных объектов (Request, Response). Сервер передает параметры запроса модулю расширения и получает сформирован-

ный Web-документ, который передается с помощью протокола HTTP обозревателю.

При многопользовательском режиме работы сервера загрузка ISAPI-модуля расширения (библиотеки DLL) происходит один раз при первом обращении. При обработке сервером последующих запросов к модулю расширения сервер использует уже загруженный экземпляр динамической библиотеки. Такой механизм взаимодействия сервера и модуля расширения обеспечивает экономию ресурсов сервера и увеличение скорости обработки запросов. Однако при возникновении ошибок в коде модуля расширения сам Web-сервер аварийно завершит работу (в отличие от интерфейса CGI, при использовании которого в случае возникновения ошибки в модуле расширения сервер может продолжать нормально функционировать). Поэтому для отладки модулей расширения, реализуемых с помощью интерфейса ISAPI, создают аналог модуля на основе интерфейса CGI. При успешной его отладке создают и отлаживают вариант модуля расширения на основе ISAPI.

ASP, PHP и IDC/HTX-страницы

ASP, PHP и IDC/HTX-страницы — это специальный тип страниц, используемых для динамического формирования на сервере Web-страниц, содержащих информацию из БД.

IDC-страница содержит алиас (псевдоним) БД, или системную запись, используемую операционной системой для связи с базой данных) БД, SQL-запрос к базе данных, идентификатор пользователя и пароль для доступа к БД. HTX-страница содержит HTML-шаблон, определяющий какую информацию и в каком формате будет иметь результирующий файл. Этот файл поддерживает все теги языка HTML и дополнительные теги для размещения информации из БД.

Активная серверная страница ASP (Active Server Page) содержит одновременно HTML-шаблон и SQL-запрос к БД. В ASP-странице используются средства языка JScript и объектная модель ASP, с помощью которых организуется доступ к БД и формируется внешний вид создаваемой Web-страницы. В ASP-страницах, также как в IDC-страницах, поддерживаются все теги языка HTML и используются дополнительные теги для размещения кода на языке JScript.

PHP-страницы разрабатываются с помощью одноименного языка обработки сценариев PHP (Personal Home Page tools — средства персональных домашних страниц). Синтаксис языка PHP напоминает смесь синтаксиса языков программирования C, JAVA и Perl.

ASP, PHP и IDC/HTX-страницы обрабатываются Web-сервером, в результате сервер генерирует Web-страницу, содержащую информацию из БД, которая отсылается обозревателю. Рассмотрим процесс формирования Web-страниц, содержащих информацию из БД.

Формирование Web-страниц

В Интернете информация находится на Web-узлах, на которых для организации взаимодействия с пользователями сети устанавливается специальное программное обеспечение, в том числе Web-сервер. В функции Web-сервера входит обработка запросов Web-обозревателей пользователей сети. В результате обработки запроса сервер формирует Web-документ, который отсылается Web-обозревателю в формате протокола HTTP.

Самостоятельно Web-сервер может отсылать готовые Web-страницы и формировать динамически Web-страницы различными способами. Для формирования динамической Web-страницы, содержащей информацию из БД, дополнительно используются модули расширения серверной части (рис. 14.1).

В связи с этим различают пассивное и активное состояния Web-сервера. Так, Web-сервер находится в пассивном состоянии, если формируемый им документ содержит статическую текстовую, графическую, мультимедийную информацию и гиперссылки. В таком документе отсутствуют средства ввода и обработки запросов к серверу.

В случае, когда на Web-странице находятся интерфейсные элементы, которые могут в ответ на реакцию пользователя обращаться с запросами к серверу, то сервер переходит в активное состояние. Для публикации БД основной интерес представляет активный Web-сервер, который реализуется с помощью программных расширений. Для создания программных расширений Web-сервера, формирующих на Web-странице содержимое БД, используются следующие средства:

- консольные exe-программы, использующие интерфейс CGI;
- exe-программы, использующие интерфейс WinCGI;
- динамические библиотеки, использующие интерфейс ISAPI ;
- динамические страницы IDC/HTX;
- активные серверные страницы ASP.

Кроме того, для организации связи программных расширений Web-сервера с БД используются современные интерфейсы доступа к данным OLE DB, ADO и ODBC. Эти интерфейсы являются промежуточным уровнем между источником данных и приложением, в качестве которого выступают программные расширения Web-сервера.

Интерфейсы OLE DB, ADO, ODBC

Современные интерфейсы доступа к источникам данным OLE DB, ADO, ODBC, внедряемые фирмой Microsoft, существенно отличаются от предшествующих интерфейсов, подобных ODBC.

Интерфейс ODBC (Open Database Connectivity – совместимость открытых баз данных) применяется операционной системой для доступа к источникам данных, как правило, к реляционным БД, использующим структурированный язык запросов SQL для организации управления данными.

Интерфейс OLE DB (Object Linking and Embedding DataBase – связывание и встраивание объектов баз данных) является более универсальной технологией для доступа к любым источникам данных через стандартный интерфейс COM (Component Object Model – объектная модель компонентов). Данные могут быть представлены в любом виде и формате (например, в качестве данных могут выступать реляционные БД, электронные таблицы, документы в rtf-формате и т. д.). В интерфейсе OLE DB используется механизм провайдеров, под которыми понимаются поставщики данных, находящиеся в надстройке над физическим форматом данных. Такие провайдеры еще называют сервис-провайдерами, благодаря им можно объединять в однотипную совокупность объекты, связанные с разнообразными источниками данных.

Кроме того, различают OLE DB-провайдер, который реализует интерфейс доступа OLE DB поверх конкретного сервис-провайдера данных. Причем поддерживается возможность многоуровневой системы OLE DB-провайдеров, когда OLE DB-провайдер может находиться поверх группы OLE DB-провайдеров или сервис-провайдеров.

Интерфейс OLE DB может использовать для доступа к источникам данных интерфейс ODBC. В этом случае применяется OLE DB-провайдер для доступа к ODBC-данным. Таким образом, интерфейс OLE DB не заменяет интерфейс ODBC, позволяет организовывать доступ к источникам данных через различные интерфейсы и в том числе ODBC.

Интерфейс ADO (ActiveX Data Objects – объекты данных ActiveX) предоставляет иерархическую модель объектов для доступа к различным OLE DB-провайдерам данных. Он характеризуется еще более высоким уровнем абстракции и базируется на интерфейсе OLE DB. Объектная модель ADO включает небольшое количество объектов, которые обеспечивают соединение с провайдером данных, создание SQL-запроса к данным, создание набора записей на основе запроса и др. Разрабатывая интерфейс ADO, фирма Microsoft предназначала его для использования в Инtranет/Интернет сетях для доступа к различным источникам данных.

Статическая публикация БД

При публикации БД на Web-страницах используются следующие способы формирования Web-страниц:

- статическая публикация Web-страниц, содержащих информацию из БД;
- динамическая публикация Web-страниц, содержащих информацию из БД.

Рассмотрим особенности формирования каждого типа страниц.

В случае *статистической* публикации Web-страницы создаются и хранятся на Web-сервере до поступления запроса пользователя на их получение (в виде файлов на жестком диске в формате Web-документа). Генерацию таких страниц может выполнять обычное Windows-приложение, имеющее доступ к БД.

Этот способ используется при публикации информации, редко обновляемой в базе данных. Обновление БД можно выполнять с требуемой периодичностью или при внесении изменений в базе данных. Такая организация публикации БД в Интернете имеет ряд преимуществ, заключающихся в получении более быстрого доступа к Web-документам, содержащим информацию из БД, и уменьшении нагрузки на сервер при обработке запросов.

Отметим, что при обработке запроса на получение Web-страницы статическим способом сервер может находиться в пассивном состоянии или в активном состоянии. Сервер находится в активном состоянии в случае, если Web-страницы содержат интерактивные элементы, которые в ответ на реакцию пользователя обращаются с запросами к серверу.

Динамическая публикация БД

Динамическая публикация используется при необходимости публиковать информацию из БД, содержимое которой часто обновляется, например, в реальном масштабе времени. Таким способом публикуется информация из БД для Интернет-магазинов и информационных систем, работающих в реальном масштабе времени, например систем продажи билетов.

При динамической публикации страницы создаются после поступления запроса пользователя на сервер. Сервер передает запрос на генерацию таких страниц программе-расширению сервера, которая формирует требуемый документ и затем сервер отсылает готовые Web-страницы обратно обозревателю. Для формирования динамических страниц используются различные средства и технологии: ASP, PHP и IDC/HTX-страницы, программы расширения сервера на основе интерфейсов CGI и ISAPI.

В случае использования ASP, PHP и IDC/HTX-страниц запрос на получение динамически формируемой Web-страницы передается специальным динамическим библиотекам, входящим в состав Web-сервера. Например, если используется Personal Web Server и публикация осуществляется средствами IDC/HTX, то применяется динамическая библиотека «httpodbc.dll». Такие библиотеки анализируют файл ASP или IDC и HTX файлы, которые используются в качестве шаблона.

Путь к файлу ASP или IDC задается в строке запроса. Сервер по расширению имени файла в строке запроса принимает решение о передаче управления требуемому модулю расширения. Если указывается exe-файл, то используется интерфейс CGI. При указании в строке запроса dll-файла применяется интерфейс ISAPI.

При реализации модулей расширения сервера в формате интерфейсов программирования CGI и ISAPI для использования шаблонов программист должен разрабатывать собственные средства. Например, в инструментальных системах быстрой разработки приложений, таких как Delphi или C++Builder, разработаны специальные компоненты, входящие в состав

библиотеки VCL, которые позволяют разрабатывать Web-приложения и автоматически генерировать Web-документ на основе шаблона, аналогичного шаблону HTML.

Отметим, что при формировании динамической Web-страницы сервер находится в активном состоянии. После отсылки страницы обозревателю сервер может перейти в пассивное состояние, если сформированная страница содержит только статическую информацию. На одном Web-сервере могут использоваться страницы, создаваемые статическим и динамическим способом.

Web-приложения

Развитие технологий Интернет вызвало переворот в индустрии программных продуктов. Среди программных средств, используемых для получения информации из сети Интернет, выделилась новая категория программ — Web-приложения, или Интернет-приложения.

Информационные системы, построенные на основе Web-приложений, характеризуются многоуровневой архитектурой и позволяют использовать все достоинства сети Интернет.

Приложения, реализующие технологию публикации БД в Интернете, составляют отдельный класс Web-приложений, под которыми понимается совокупность Web-страниц, клиентских и серверных сценариев, расположенных на одном или нескольких компьютерах и выполняемых в рамках одной информационной системы (целевой задачи). Для разработки Web-приложений могут использоваться различные комбинации рассмотренных технологий. Web-приложения, публикующие содержимое БД, по своей архитектуре и организации работы опираются на принципы, заложенные в многоуровневые клиент-серверные приложения. Однако Web-приложения имеют свои особенности функционирования, заключающиеся в организации работы сети Интернет. Разработчику Web-приложения требуется учитывать следующие вопросы:

- совместимость Web-обозревателей;
- разграничение доступа и обеспечение безопасности данных;
- надежность линий связи.

Web-приложения, публикующие БД на Web-страницах, выполняются на стороне сервера. Сервер обрабатывает запросы обозревателя. Запросы к БД сервер передает серверу приложений или серверу баз данных. Обработав запрос, сервер БД передает нужные данные Web-серверу, который формирует Web-документ и отсылает его обозревателю.

Функции обозревателя заключаются в отображении Web-страниц, сгенерированных сервером или модулями расширения, и отправке запросов пользователя Web-приложению. Обозреватель является связующим звеном между пользователем и Web-приложением.

Протоколы передачи гипертекста

В сети Интернет могут использоваться различные платформы и серверы: для операционной системы Windows NT и Windows 2000 – это Microsoft Internet Information Server (MS IIS), для UNIX систем – Apache Server (этот сервер распространяется в исходных кодах, поэтому после перекомпиляции может выполняться и в других операционных системах), а для Windows 98 – Microsoft Personal Web Server.

Запросы от обозревателя поступают на сервер в соответствии с установленным протоколом обмена, например, HTTP (HyperText Transfer Protocol – протокол передачи гипертекста). В сети Интернет существует более десятка протоколов. Протокол HTTP является одним из самых распространенных протоколов и предназначен для передачи данных различных форматов между обозревателем и сервером. Соединение между компьютером-отправителем и компьютером-получателем осуществляется с помощью протокола низкого уровня TCP/IP (Transfer Control Protocol/Internet Protocol – протокол управления передачей/Интернет протокол).

Протокол TCP/IP представляет собой универсальный независимый от платформы протокол передачи данных по сети. За физическую передачу данных отвечает протокол IP. TCP представляет собой протокол более высокого уровня. Он разбивает в компьютере-отправителе файлы на пакеты, добавляет в каждый из них адрес получателя и их порядковый номер пакета в группе пакетов. В компьютере-получателе протокол TCP собирает файлы из пакетов, проверяет их целостность и может посыпать запрос на повторную передачу.

При отправке серверу HTTP-запроса на формирование динамической страницы, содержащей отчет из БД, сервер включает в работу модули расширения. Взаимодействие Web-сервера и модуля расширения основывается на интерфейсах CGI или API. Модуль расширения получает запрос и обрабатывает его, формируя ответ, обычно Web-страницу.

Универсальный указатель ресурсов

Обозреватель формирует запрос на получение нужной страницы с помощью универсального указателя ресурса – URL (Universal Resource Locator). Для загрузки требуемой страницы в окне обозревателя указывается строка адреса. Загрузку требуемого документа можно осуществить и из Web-документа с помощью специальных тегов.

Приведем структуру URL для протокола HTTP:

`http://<хост>/<порт>:<путь>?<поиск>`

Здесь:

- http – имя протокола, используемого для доступа к ресурсу

- <хост> — имя домена, используемое для поиска требуемого Web-узла в Интернет
 - <порт> — номер порта, который задает номер логического канала связи в Интернете
 - <путь> — задает локальный путь к файлу
 - <поиск> — дополнительные параметры запроса
- В обозревателях используют следующие два типа адресов указателей:
- *Абсолютные указатели* задают адрес, полностью определяющий компьютер, каталог и файл. Такой адрес называется *абсолютным*. В отличие от относительных адресов абсолютные адреса могут ссылаться на файлы, расположенные на других компьютерах;
 - *Относительные указатели* используются для получения доступа к файлу, расположенному на том же компьютере, что и документ, в котором находится указатель на этот файл. Например, если обозреватель загрузил страницу, находящуюся по адресу “<http://myweb>”, то при задании относительного адреса “/my” реальный адрес будет “<http://myweb/my>”. Относительные адреса удобны в использовании в пределах одного компьютера.

Основу Web-приложений составляет совокупность Web-документов и различных шаблонов, использующих формат языка HTML или его обобщений. В связи с этим для разработки Web-приложений прежде всего необходимо знание особенностей языка HTML.

В следующем разделе рассматриваются конструкции языка HTML, которые используются для разработки Web-приложений, публикующих БД в Интернете.

14.2. Состав и теги HTML-документа

HTML-документ есть разновидность Web-документа и в общем случае представляет собой текстовый документ (обычный ASCII-файл), который может быть отредактирован в любом текстовом редакторе. Он состоит из текстовых данных, обрамленных специальными управляющими операторами (тегами). Теги несут в себе служебную информацию для обозревателя и могут задавать различные режимы форматирования. По сути дела язык HTML является языком разметки документов, используемых в Интернете.

Теги заключаются в угловые скобки. Обозреватель не отображает теги, они несут для него информацию по форматированию текста. Например, начальным тегом документа является следующий тег:

<HTML>

Теги могут быть парными и непарными. Парные теги делятся на открывающий и закрывающий. Пара таких тегов называется *контейнером*.

Например, весь HTML-документ обрамляется следующими тегами:

```
<HTML>
...
<!--Это комментарий, который обозреватель не отображает. -->
Текст HTML-документа
...
</HTML>
```

Здесь `<HTML>` – открывающий тег, `</HTML>` – закрывающий тег, между ними могут находиться некоторые текстовые данные.

`<!--Это комментарий, который обозреватель не отображает.-->` – это пример непарного тега, который является комментарием и обозревателем не отображается.

HTML-документ состоит из двух частей — заголовка и тела документа.

Заголовок документа является не обязательным элементом, он может использоваться для того, чтобы задать название документа, определить отношения между несколькими документами и т. д. Заголовок и тело документа обрамляются специальными тегами. В теле документа находится текст документа и различные управляющие теги.

По функциональному назначению различают следующие основные группы тегов:

- структурные теги, отвечающие за формирование основных разделов (частей) HTML-документа;
- теги форматирования текста, управляющие выводом текста в окно обозревателя;
- табличные теги, используемые для создания простейших таблиц;
- теги кадров, служащие для разделения окна обозревателя на изолированные области вывода данных;
- теги форм, служащие для поддержания диалога между HTML-страницей и пользователем;
- теги графических изображений, позволяющие выводить изображения в окне обозревателя;
- ссылочные теги, формирующие ссылки на части документа или другие HTML-документы;
- теги-дескрипторы внедряемых объектов, позволяющие внедрять аплиты, дополнительные модули, элементы управления ActiveX и другие объекты.

Существует большое количество различных тегов. Поясним использование основных тегов, входящих в указанные группы, которые могут понадобиться для создания собственного Web-приложения, и приведем примеры их практического использования.

Структурные теги

Структурные теги отвечают за выделение основных разделов (частей) HTML-документа. К ним относятся теги границ документа (`<HTML> ... </HTML>`), теги заголовка документа (`<HEAD> ... </HEAD>`), теги тела документа (`<BODY> ... </BODY>`).

Тег `<HTML>` контейнерного типа, указывает начало HTML-документа, а тег `</HTML>` указывается в последней строке документа. Эти теги обозначают, что находящиеся между ними строки представляют единый HTML-документ, а не обычновенный файл в формате ASCII.

Замечание.

Хотя большинство современных обозревателей могут опознать документ и не содержащий тегов `<HTML>` и `</HTML>`, но, учитывая разнообразные области применения HTML (электронная почта и др.), их употребление желательно.

Внутри тегов `<HEAD> ... </HEAD>`, определяющих заголовок документа, могут находиться другие теги, задающие название документа и другую общую информацию. Например, название документа, которое обычно показывается в заголовке окна обозревателя, записывают между тегами `<TITLE>` и `</TITLE>` в виде текстовой строки.

Тело документа задается с помощью тегов `<BODY> ... </BODY>` в следующем формате:

```
<BODY атрибуты>
... содержимое документа...
</BODY>
```

Здесь атрибуты не обязательны и определяют внешний вид документа; содержимое документа – любые допустимые элементы HTML-документа.

Например, HTML-документ, содержащий заголовок и одну строку текста, имеет следующий вид:

```
<HTML>
<HEAD><!--Это заголовок документа. -->
<title> Это заголовок простого HTML-документа! </title>
</HEAD>

<BODY><!--Это тело документа. -->
Это текст простого HTML-документа!
</BODY>

</HTML>
```

Приведенный пример представляет собой описание HTML-документа с минимальным набором тегов.

Теги форматирования текста

Теги форматирования текста используются для того, чтобы HTML-документ был более наглядным и читаемым. В этом разделе приведем некоторые наиболее часто используемые из таких тегов.

Разделение текста на абзацы осуществляется с помощью тега контейнерного типа `<P ALIGN=атрибут>...</P>`. Применение этого тега позволяет разделять текст на абзацы независимо от установок параметров окна обозревателя. В теге абзаца атрибут `ALIGN` может принимать значения `LEFT`, `CENTER` и `RIGHT`, задающие выравнивание текста по левому краю, по центру, и по правому краю соответственно. По умолчанию (при отсутствии атрибута) выравнивание текста осуществляется по левому краю.

Перевод каретки (каретка указывает на текущую позицию, в которую будет производится следующий вывод текста) на следующую строку в любом месте HTML-документа осуществляется тегом разрыва строки `
`. В отличие от тега абзаца тег `
` не пропускает строку.

Для повышения читабельности HTML-документа используются заголовки. Тег `<H1>` задания заголовка является контейнерным и имеет открывающий (`<H1 ALIGN=атрибут>`) и закрывающий (`</H1>`) теги. HTML располагает шестью уровнями (относительными размерами шрифта) заголовков: `H1` (самый верхний), `H2`, `H3`, `H4`, `H5` и `H6` (самый нижний). Например, задание заголовка со вторым уровнем размера шрифта имеет вид:

`<H2> Текст из файла </H2>`.

Выравнивание заголовков, как и в случае выравнивания абзацев, осуществляется с помощью атрибута `ALIGN`.

Тег `<HR>` позволяет провести рельефную горизонтальную линию в окне большинства программ просмотра, он не требует закрывающего тега. При указании этого тега до и после линии автоматически вставляется пустая строка.

Тег `<I>` является контейнерным и используется для выделения текста курсивом. Например:

`<I> Выделенный текст </I>`

Тег `` является контейнерным и используется для выделения фрагментов текста полужирным шрифтом. Например:

`Этот текст будет выделен полужирным шрифтом!`

Тег `<TT>` является контейнерным и используется для вывода текста шрифтом фиксированной ширины. Например:

`<TT> Этот текст выведен с использованием шрифта фиксированной ширины! </TT>`

Тег **<U>** является контейнерным и используется для подчеркивания текста. Например:

```
<U>Подчеркнутый текст</U>
```

Тег **** является контейнерным и служит для задания различных атрибутов шрифта текста. После открывающего тега обязательно указание атрибутов, без которых элемент не оказывает влияния на текст, помещенный в контейнер. Рассмотрим использование некоторых атрибутов рассматриваемого тега.

Атрибут **FACE** используется для задания гарнитуры шрифта, которым обозреватель отображает соответствующий текст. При отсутствии шрифта нужной гарнитуры обозреватель использует шрифт с гарнитурой, установленной по умолчанию. Например:

```
<FONT FACE="Arial"> Этот текст выводится с использованием шрифта Arial. </FONT>
```

Атрибут **SIZE** используется для указания размера шрифта в условных единицах от 1 до 7. Например:

```
<FONT SIZE=1> Текст выводится с использованием размера шрифта = 1
</FONT>
```

Пример простейшего HTML-документа, в котором использовано форматирование текста:

```
<HTML>
<HEAD> <!--Это заголовок документа. -->
<title> Это заголовок HTML-документа! </title>
</HEAD>
<body> <!--Это тело документа. -->
<H2> Текст из файла </H2>
<BR>
<p ALIGN=CENTER>
<FONT FACE="Arial">
<I> Выделенный текст, выведенный с использованием шрифта Arial</I>
</FONT>
<BR>
<FONT SIZE=2>
<B>Этот текст будет выделен полужирным шрифтом и имеет размер 2!</B>
<BR>
<TT> Этот текст выведен с использованием шрифта фиксированной
ширины! </TT>
</p>
</body>
</HTML>
```

Вид описанного в приведенном примере HTML-документа в окне обозревателя Microsoft Internet Explorer показан на рис. 14.2.

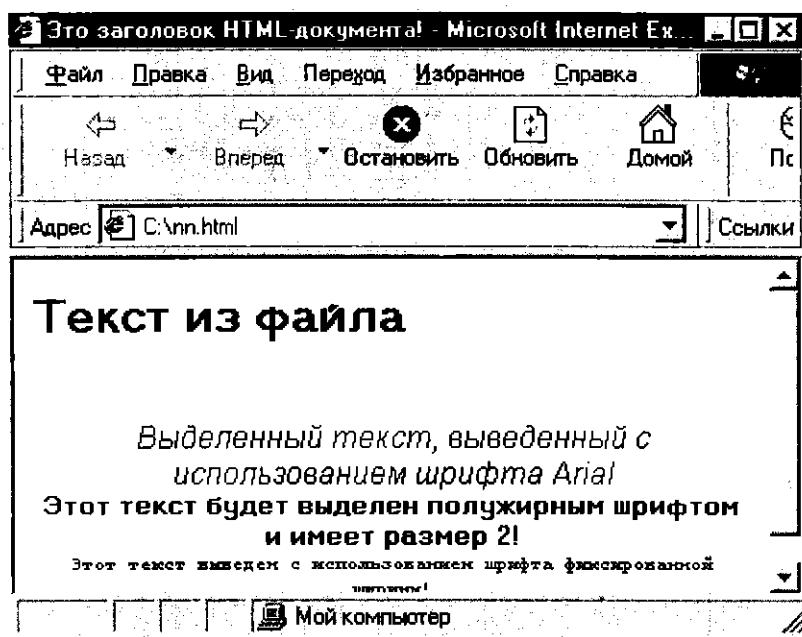


Рис. 14.2. Вид документа с форматированием текста в окне обозревателя

В приведенном примере использованы теги, позволяющие задавать различные форматы вывода текста.

Табличные теги

Для создания таблиц используется табличный тег <TABLE>, являющийся контейнерным тегом. В этом теге-контейнере размещается содержимое таблицы. В теге <TABLE> могут использоваться следующие атрибуты:

- **BORDER** – для определения рамки таблицы. Ширина рамки устанавливается в пикселях, например BORDER=2 (по умолчанию 1);
 - **ALIGN** – для выравнивания таблицы в окне обозревателя. Имеет значения LEFT (по умолчанию), CENTER и RIGHT.
- Структура таблицы определяется с помощью следующих тегов-контейнеров:
- <TH> – задает заголовки столбцов (строк), которые выделяются полужирным шрифтом и центрируются в своих ячейках;
 - <TD> – задает данные в ячейках.

Пример. Задание таблицы, содержащей 3 строки и 3 столбца.

```
<TABLE ALIGN=LEFT BORDER=1>
<TR> <TH>Заголовок 1 столбца</TH>
<TH> Заголовок 2 столбца</TH>
<TH> Заголовок 3 столбца </TH></TR>
<TR> <TD>Данные 2 стр. и 1 столб.</TD>
<TD> Данные 2стр. и 2 столб.</TD>
<TD> Данные 2стр. и 3 столб.</TD> </TR>
<TR> <TD> Данные 3 стр. и 1 столб.</TD>
<TD> Данные 3 стр. и 2 столб.</TD>
<TD> Данные 3 стр. и 3 столб.</TD> </TR> </TABLE>
```

Этот пример задает таблицу, обрамленную рамкой, содержащую 3 строки и 3 столбца.

Аналогичным образом могут задаваться таблицы с произвольным числом строк и столбцов.

Теги определения кадров

Под кадром (фреймом – от англ. термина FRAME) понимается некоторая отдельная часть окна обозревателя. Кадры определяются внутри кадровой структуры (блока определений кадров), именуемой FRAMESET.

Для определения блока кадров используется контейнерный тег **<FRAMESET>**, имеющий следующий формат

```
<FRAMESET атрибут1=значение... атрибутN=значение >
...Теги кадров ...
</FRAMESET>.
```

Внутри этого тега-контейнера могут располагаться только теги **<FRAME>**, определяющие одиночные кадры, или другие контейнеры **<FRAMESET>**. Рассматриваемый тег замещает тег **<BODY>**. HTML-документ, имеющий структуру FRAMESET, не должен иметь тег **<BODY>**, иначе кадровая структура будет игнорирована.

Рассмотрим применение важных атрибутов ROWS и COLS тега **<FRAMESET>**, определяющих строки и столбцы как отдельные кадры в кадровой структуре. В HTML-документе можно определить любое число строк и столбцов как кадров. При этом необходимо задать значение для хотя бы одного из атрибутов ROWS или COLS. Тег **<FRAMESET>** с атрибутами ROWS и COLS имеет следующий формат:

```
<FRAMESET ROWS="список значений" COLS=" список_значений " >
</FRAMESET>
```

Здесь список_значений содержит отделенные через запятую значения параметров, определяющих размеры кадров в пикселях, процентах или относительных единицах.

Следует учитывать, что кадр не может быть единственным. Для того чтобы обозреватель отобразил кадровую структуру, необходимо задать в списке значений параметров атрибутов ROWS или COLS более одного значения. Например:

```
<FRAMESET ROWS="50,50,100,100">  
...  
<FRAMESET COLS="50%,50%">
```

Здесь первая строка определяет набор кадров из четырех строк, высота которых составляет 50,50,100,100 пикселов соответственно, вторая строка определяет набор кадров из двух столбцов.

Высоту строки в пикселях задавать нежелательно, так как при этом не учитывается, что окна обозревателей могут иметь различную величину. В абсолютных единицах можно указывать размеры кадра лишь для небольших изображений, в остальных случаях лучше пользоваться относительными величинами, например:

```
<FRAMESET ROWS="25%,50%,25%">
```

Приведенный код создает три строки кадров высотой в 25%, 50% и 25% от высоты окна обозревателя.

Замечание.

Относительную точность указания высоты кадров в процентах можно не волноваться: если сумма не равна 100%, масштаб кадров будет пропорционально изменен.

Задание параметров кадров в относительных единицах выглядит следующим образом:

```
<FRAMESET COLS="*,2*,3*">
```

Здесь символ * означает пропорциональное деление окна программы просмотра. Таким образом, окно будет разделено на три вертикальных кадра, первый из которых будет иметь ширину в 1/6, второй — в 2/6 (или 1/3) и третий — в 3/6 (или 1/2) от ширины окна обозревателя. Единица при указании относительных значений может быть опущена.

Указание значений атрибутов ROWS и COLS может быть и смешанным, например:

```
<FRAMESET COLS="100,25%,*,2*">
```

Здесь первому кадру будет присвоено абсолютное значение в 100 пикселов по ширине, второму — 25% от ширины окна. Оставшееся пространство делится между третьим и четвертым кадрами в пропорции 1/3 к 2/3.

Приоритеты в определении значений атрибутов следующие: в первую очередь слева направо отводится место для кадра с абсолютным значением, затем — для кадра со значением в процентах, а в последнюю очередь — для кадров с относительными величинами.

Замечания.

При использовании абсолютных значений в атрибутах ROWS и COLS не следует задавать такие кадры большими — они должны поместиться в окне обозревателя любого размера. Совместно с такими кадрами для лучшей балансировки рекомендуется использовать хотя бы один кадр, определенный в процентах или относительных величинах. Каждому кадру соответствует свой HTML-документ, содержащий отображаемую в нем информацию.

Для разбивки окна обозревателя хотя бы на два кадра необходимо создать 3 файла (один файл используются под основной документ, в котором задается кадровая структура документа, и еще два файла используются под каждый кадр).

Для пояснения изложенного материала рассмотрим код документа main.htm, в котором осуществляется разбивка документа на 3 кадра:

```
<HTML>
<HEAD>
</HEAD>

<FRAMESET ROWS="50%,50%">
<FRAME SRC="file1.htm">
<FRAMESET COLS="25%,75%">
<FRAME SRC="file21.htm"> <FRAME SRC="file22.htm">
</FRAMESET>
</FRAMESET> <NOFRAMES>
```

Ваш обозреватель не поддерживает кадровую структуру.

```
</NOFRAMES> </HTML>
```

В файле file1.htm содержится следующая информация:

```
<Font Size=10>Первый кадр</Font>
```

В файле file21.htm содержится следующая информация:

```
<Font Size=2>Второй кадр, первый подкадр</Font>
```

В файле file22.htm содержится следующая информация:

```
<Font Size=3>Второй кадр, второй подкадр</Font>
```

Внешний вид окна обозревателя Netscape Navigator, отображающего содержимое файла main.html, приведен на Рис. 14.3.



Рис. 14.3. Вид окна обозревателя, содержащего файл main.htm

Как видно из рисунка, документ состоит из трех кадров. Верхний кадр занимает половину окна обозревателя по высоте. Нижняя половина поделена еще на два кадра. Левый нижний кадр (первый подкадр), на долю которого приходится 25% ширины окна обозревателя. Нижний правый кадр (второй подкадр) занимает 75% от ширины окна обозревателя. Такая разбивка на кадры реализуется на основе включения в набор кадров тега <FRAMESET> еще одного набора кадров.

Теги создания форм

Формы HTML являются основным средством организации интерактивного взаимодействия в Интернете при разработки Web-приложений. Они служат для пересылки данных от пользователя к Web-серверу. С помощью элементов HTML-документа можно создавать простые формы, предполагающие ответы типа "да" и "нет", а также разрабатывать сложные формы для заказов или для того, чтобы получить от пользователей какие-либо комментарии и пожелания, создавать требуемые интерфейсные элементы, фиксировать выбор параметра пользователем и т. д.

В HTML-документе формы задаются с помощью контейнерного тега <FORM> и состоят из нескольких полей ввода (ограниченная область на экране обозревателя), которые обозреватель отображает в виде графических

элементов управления: кнопок выбора, переключателей, строк ввода текста, управляющих кнопок и т. д. Для создания полей ввода предназначены теги **<INPUT>**, **<SELECT>**, **<TEXTAREA>**, которые задаются внутри тега **<FORM>**. Рассмотрим коротко теги, используемые для создания форм и полей ввода на форме.

Тег **<FORM>** в общем виде записывается в следующем формате:

```
<FORM ACTION="URL" METHOD=метод_передачи ENCTYPE=MIME-тип>
    содержимое формы
</FORM>
```

Параметр **ACTION** является обязательным. В качестве его значения задается адрес URL программы-сценария (модуля расширения сервера на базе интерфейса CGI, ISAPI), которая будет обрабатывать извлеченную из формы информацию.

Параметр **METHOD** определяет метод пересылки данных формы от обозревателя к Web-серверу. Если параметр имеет значение **GET**, то данные формы передаются в составе запроса URL, будучи присоединенными к нему справа от символа "?" в виде пар "переменная=значение", разделенных символом "&". Например, первая строка запроса может иметь следующий вид:

```
GET /bhv.ru/cgi-bin/prog1.cgi?name=stive&format=HTML HTTP/1.0
```

Web-сервер после получения запроса присваивает переменной среды **QUERY_STRING** значение строки запроса и вызывает программу-сценарий, указанную в начальной части строки запроса. Эта программа может обратиться к переменной среды **QUERY_STRING** для обработки размещенных в ней данных.

Если параметр **METHOD** имеет значение **POST**, то данные формы пересылаются Web-серверу в теле запроса, который передает их, например, в программу CGI через стандартный ввод.

Значение параметра **ENCTYPE** определяет формат кодирования данных (или медиа-тип) содержимого формы при передаче их от обозревателя к Web-серверу. Кодирование данных выполняется для предотвращения их искажения при передаче.

Рассмотрим использование тегов-дескрипторов **<TEXTAREA>**, **<SELECT>** и **<INPUT>**, применяемых в HTML для создания интерфейсных объектов внутри формы (тег-дескриптор – это тег, который может использоваться только внутри тега **<FORM>**).

Замечания.

Использование тегов-дескрипторов вне форм поддерживается в обозревателе Internet Explorer. Netscape Navigator требует использовать теги-дескрипторы только внутри тега-контейнера **<FORM>**.

Теги-дескрипторы имеют следующее назначение:

- <TEXTAREA> позволяет вводить многострочную текстовую информацию;
- <SELECT> используется для выбора нужной строки в окне с полосой прокрутки либо в раскрывающемся меню;
- <INPUT> является многоцелевым тегом и позволяет вводить текстовую информацию в виде строки, создавать интерфейсные объекты в виде кнопок по установке и сбросу флагов (check boxes) и переключателей (radio buttons), обычные кнопки и другие объекты.

Тег <SELECT>

Тег <SELECT> является контейнерным и позволяет организовать внутри формы выбор одной опции (параметра) из набора нескольких кнопок без использования полей выбора и переключателей. Тег <SELECT> позволяет выполнить компактное представление элементов выбора в виде раскрывающегося списка или списка прокрутки.

Тег <SELECT> записывается в следующем формате:

```
<SELECT NAME=имя_поля SIZE=число MULTIPLE>
  элементы меню или списка
</SELECT>
```

Здесь: параметр SIZE определяет количество видимых элементов выбора, при SIZE=1 используется раскрывающееся меню, при SIZE="число" большем 1 используется список прокрутки, для которого "число" определяет количество видимых элементов; задание параметра MULTIPLE позволяет выполнять выбор из меню или списка одновременно несколько элементов.

Элементы меню в теге <SELECT> задаются с помощью тега <OPTION>, записываемого в следующем формате:

```
<OPTION SELECTED VALUE=значение>текст
```

Атрибут SELECTED задает выбор элемента по умолчанию. Параметр VALUE содержит значение, пересылаемое серверу, если данный элемент выбран в меню или списке. Текст, находящийся в конце тега, выводится на экран обозревателя на месте элемента в списке.

Тег <TEXTAREA>

Тег <TEXTAREA> является контейнерным и используется для создания внутри формы поля ввода многострочного текста. Это поле может отображаться в виде ограниченной прямоугольной области с горизонтальной и вертикальной полосами прокрутки. Тег имеет следующий формат:

```
<TEXTAREA NAME=имя ROWS=число COLS= число >
  текст по умолчанию
</TEXTAREA>
```

Атрибуты ROWS и COLS определяют число строк и число столбцов видимого текста соответственно. Между открывающим и закрывающим тегами может быть размещен текст, отображаемый в поле ввода по умолчанию.

Тег <INPUT>

Тег <INPUT> предназначен для генерирования внутри формы полей для ввода текста, пароля, имени файла и задания различных кнопок. Тег имеет следующий формат:

```
<INPUT TYPE=тип_поля_ввода NAME=имя_поля_ввода дополнительные_параметры>
```

Первые два параметра являются обязательными. Параметр TYPE определяет тип поля ввода: кнопка выбора, кнопка передачи и др. Параметр NAME задает имя поля, которое используется как идентификатор передаваемого Web-серверу значения. Состав дополнительных параметров зависит от значения параметра TYPE, определяющего типа поля.

Атрибут TYPE тега <INPUT> может принимать следующие значения:

- **TEXT** – является значением по умолчанию. В этом случае создается интерфейсный элемент в виде одной строки для ввода данных. Для этого типа поля ввода наиболее часто используется атрибут VALUE, которое предоставляет доступ к данным находящимся внутри этого поля;
- **PASSWORD** – аналогичен предыдущему типу, но заменяет вводимые символы пароля звездочками;
- **CHECKBOX** – выводит поле для установки флагов в виде ограниченной области, в котором может быть произведен выбор сразу несколько опций из числа предложенных. В этом поле может использоваться атрибут CHECKED, который определяет установленный по умолчанию флаг;
- **RADIO** – аналогичен предыдущему типу поля, но имеет отличный внешний вид (аналогичный интерфейсному виду Radio-кнопки – графическому элементу операционной системы Windows) и не позволяет отменять сделанный выбор;
- **RESET** – позволяет создать кнопку для обновления формы;
- **SUBMIT** – используется для создания кнопки, по нажатию которой введенные данные отправляются на сервер для обработки программой-сценарием.

Например, создание различных элементов тега INPUT и других тегов формы для отправки информации на сервер выполняется с помощью следующего кода:

```
<FORM>  
Ведите ваше имя:  
<INPUT TYPE="text" NAME="My" Value="Имя" SIZE="10" MAXLENGTH="20">  
Ведите пароль:  
<INPUT TYPE="password" NAME="pass1" Size="30" MAXLENGTH="30">
```

```
<BR>
Выберите тип действия: <BR>
<INPUT TYPE="checkbox" NAME="mycheckbox1" VALUE="Параметр1"
CHECKED>
Действие 1
<INPUT TYPE="checkbox" NAME="mycheckbox2" VALUE=" Параметр2">
Действие 2
<BR>
<INPUT TYPE="radio" NAME="my1" VALUE=" Параметр2">
Действие 3
<INPUT TYPE="radio" NAME="my2" VALUE=" Параметр3">
Действие 4
<BR>
<TEXTAREA NAME="My_textarea" ROWS=4 COLS=40>
Справочный текст, внутри которого <BR>
можно ввести собственные данные <BR>
Эта форма посыпает данные серверу
</TEXTAREA> <BR>
<SELECT NAME="menu">
<OPTION SELECTED VALUE="Действие1"> Действие1
<OPTION VALUE=" Действие2"> Действие2
<OPTION VALUE=" Действие3"> Действие3
</SELECT>
<INPUT TYPE="reset" VALUE="Reset ">
<BR>
<INPUT TYPE="submit" VALUE="Послать данные!">
</FORM>
```

На Рис. 14.4 приведено содержимое окна обозревателя, содержащее текст примера с элементами различных типов тегов INPUT, SELECT и TEXTAREA.

В этом примере у элемента “текстовое поле” использованы дополнительные атрибуты: SIZE="10" определяет размер рамки для размещения 10 символов, а атрибут MAXLENGTH="20" максимальный размер внутреннего представления текстовой строки в символах. Атрибут NAME у всех типов используется для получения доступа к заданному элементу HTML-документа.

Графические теги

В HTML-документе используются два формата графических изображений: GIF и JPEG. Эти форматы используют большинство современных обозревателей.



Рис. 14.4. Окно обозревателя с текстом примера

Для использования изображения на HTML-странице применяется тег **IMG** следующего формата:

```
<IMG атрибут1=значение ... атрибутN=значение >
```

Рисунок отображается в том месте документа, где находится тег ****. Отметим применение некоторых наиболее важных атрибутов тега ****.

Атрибут **SRC** используется для указания URL-адреса файла, содержащего рисунок. Например:

```
<IMG SRC="my.gif">
```

Атрибуты **HEIGHT** и **WIDTH** позволяют устанавливать размер изображения в пикселях соответственно по высоте и ширине. Например:

```
<IMG SRC="my.gif" HEIGHT=200 WIDTH=100 >
```

Замечание.

Использовать эти атрибуты нужно весьма аккуратно, так как при изменении масштаба изображения программа просмотра может сильно искажить рисунок.

Атрибут ALT позволяет выводить краткое текстовое описание вместо рисунка, в случае если пользователь установил отмену загрузки изображений. Если изображения выводится, то это описание отображается на месте рисунка до начала его загрузки. Например:

```
<IMG SRC="my.gif" ALT="Описание рисунка">
```

Атрибут BORDER используется для установления толщины рамки в пикселях, которая выделяет рисунок.

Теги задания ссылок

Одним из важных элементов в HTML-документе является ссылка. Она состоит из *указателя* (anchor) и *адресной части ссылки* (URL-адрес). Указатель ссылки отображается обозревателем в HTML-документе. При выборе пользователем указателя ссылки обозреватель загружает HTML-документ, адрес которого определяется URL-адресом. В качестве указателя ссылки может выступать текст или изображение. Обычно обозреватель отображает указатель ссылки в виде подчеркнутого прямой линией текста.

Ссылки задаются с помощью тега контейнерного типа следующего формата:

```
<A HREF="URL-адрес">Текст указателя или Теги задающие  
изображение</A>
```

Атрибут HREF тега **<A>** задает URL-адрес HTML-документа, который отображается в окне обозревателя при выборе ссылки.

Информация находящаяся внутри контейнера, может содержать текст указателя или специальные теги, задающие изображения. Например:

```
<A HREF="my.html">Для отображения файла my.html выберите этот  
текст </A>
```

Для создания графической ссылки следует в контейнер тега ссылки включить тег, определяющий изображение, например:

```
<A HREF="my.html"><IMG SRC="рисунок.gif"></A> .
```

В случае необходимости ссылаться на разные части текущего документа используются *внутренние ссылки*. Для задания внутренней ссылки нужно в атрибуте HREF тега **<A>** поместить имя указателя со специальным префиксом (#):

```
<A HREF="#Имя"> Текст внутри ссылки </A>
```

где Имя – специальное имя, которое используется для организации связи с фрагментом текста; Текст внутри ссылки – текст указателя ссылки, отображаемый в окне обозревателя.

Например:

```
<A HREF="#Next">Переход на следующий фрагмент текста</A>
```

Для указания места перехода по внутренней ссылке используют следующий тег:

```
<A NAME=Имя>Фрагмент текста, на который осуществляется переход</A>
```

Здесь: Имя — имя указателя, которое входит в атрибут HREF тега внутренней ссылки;

Например:

```
<A HREF="#Go">Текст для перехода по внутренней ссылке </A>
```

...

```
<A NAME=Go>Фрагмент текста, к которому осуществляется переход  
по внутренней ссылке </A>
```

Среди технологий Интернета становится популярным расширяемый язык разметки XML. Этот язык имеет средства структурированного представления данных и доступа к ним, на основе принципов, применяемых в СУБД. Ввиду этого он может использоваться для публикации сильно структурированных данных (а именно, БД) па HTML-страницах. В следующем разделе рассмотрим особенности использования языка XML.

14.3. Особенности XML-документа

Расширяемый язык разметки XML (eXtensible Markup Language) представляет собой развитие языка HTML и по сравнению с ним обеспечивает ряд дополнительных возможностей. В их числе можно выделить следующее:

- возможность подготовки и настройки XML-документов со сложной структурой;
- использование информационных объектов (entity) для работы с группами данных;
- описание типа документа DTD (Document Type Definition);
- возможность контроля документа на предмет его правильности;
- использование XML-формата для хранения структурированных однотипных данных (раннее для этой цели использовали только БД).

Отметим, что XML принадлежит подмножеству стандартного обобщенного языка разметки SGML (Standard Generalized Markup Language). Последний язык является прародителем языков разметки. Язык SGML, хотя и предоставляет более развитые возможности по сравнению с XML, но его применение требует больших затрат времени на подготовку и обработку

соответствующего SGML-документа. Поэтому он не получил такого широкого распространения, как XML.

Главное отличие XML от HTML заключается в том, что с помощью XML выполняется не только наполнение создаваемого документа содержанием с указанием разметки, а в основном определяется структура документа и типы хранимых в нем данных. С использованием XML обеспечиваются средства стандартизованного представления данных, тем самым облегчается задача использования (импорта) данных из других источников.

Документы XML можно разделить на правильно построенные (*well-formed*) и действительные (*valid*). Правильно построенные документы XML удовлетворяют спецификации XML, но не имеют определения типа документа DTD (Document Type Definition). Действительные документы XML, в отличие от правильно построенных, содержат также определение типа документа DTD (определяющее типы используемых данных и возможную структуру документа) илизываются на него.

Спецификация XML поддерживается рядом современных программам просмотра документов Web. В частности, возможность просмотра XML-документов обеспечивается Internet Explorer, начиная с версии 4.0, а также Netscape Navigator 5.0 и выше.

По сравнению с документом HTML при описании XML-документа накладываются следующие дополнительные ограничения:

- каждый открывающий тег должен иметь закрывающий тег;
- пары открывающих и закрывающих тегов могут вкладываться друг в друга, но не допускаются пересечения открывающего и закрывающего тегов;
- при определении элементов и атрибутов учитывается регистр символов;
- все значения атрибутов должны указываться в кавычках одинарных (' ') или двойных (" ") .

Рассмотрим более подробно составляющие XML-документа, их назначение, объявления и примеры использования.

Составляющие документа

В XML-документе обычно используются следующие составляющие: элементы, атрибуты, информационные объекты и комментарии. Элементы составляют части документа. Объявление элемента задается в следующем формате:

<!ELEMENT имя содержание>

Объявление элементов может размещаться внутри документа или в определении типа документа DTD. В DTD можно объявить произвольное число элементов, при этом не все из них должны использоваться в документе XML. Отдельные элементы могут содержать другие элементы. Например, для указания, что элемент сотрудник (*employee*) содержит элементы фамилия

(surname) и должность (post), нужно поместить в DTD определения элементов в следующем порядке:

```
<!ELEMENT surname(#PCDATA)>
<!ELEMENT post(#PCDATA)>
<!ELEMENT employee(surname, post)>
```

Здесь строка #PCDATA является атрибутом и указывает, что содержание соответствующего элемента могут составлять символьные данные.

При необходимости можно указать, что элемент в содержании документа необязателен или может использоваться несколько раз. Основные символы, используемые при описании элемента, приведены в таб. 14.1.

Таблица 14.1.

Основные символы для описания элемента

Символ	Имя символа	Назначение
()	Круглые скобки	Содержит альтернативные элементы или элементы списка
	Вертикальная линия	Разделение альтернативных элементов
?	Знак вопроса	Элемент может отсутствовать или использоваться один раз
,	Запятая	Разделение элементов в списке
*	Звездочка	Элемент может отсутствовать или использоваться любое число раз
+	Плюс	Элемент должен использоваться не менее одного раза

Например, в строке

```
<!ELEMENT employee(surname, post+)>
```

знак “+” указывает на то, что в составе элемента employee элемент post должен присутствовать не менее одного раза.

Атрибуты служат для уточнения характеристик элементов, вида содержащейся в них информации и порядка ее размещения в них. Атрибуты можно указывать в строке объявления элемента (как показано выше) или в DTD в задаваемом с помощью тега <!ATTLIST> списке атрибутов. Для действительного документа XML объявление атрибута в DTD задается в следующем формате:

```
<!ATTLIST имя_элемента имя_атрибута тип значение_по_умолчанию>
```

Здесь имя_элемента указывает элемент, для которого задается атрибут; имя_атрибута определяет имя атрибута; тип указывает один из 10 возмож-

ных типов атрибутов; значение_по_умолчанию определяет значение атрибута элемента, используемое по умолчанию.

Атрибут элемента может быть строковым, маркированным или перечислимым и иметь следующие типы и назначение:

- **CDATA** – символьные данные;
- **Enumerated** – набор возможных значений;
- **ID** – уникальный идентификационный номер;
- **IDREF** – указатель на элемент с заданным значением ID;
- **IDREFS** – указатель на несколько элементов в документе с помощью списка ID, разделенного пробелами;
- **ENTITY** – имя внешнего информационного объекта;
- **ENTITIES** – указатель на несколько внешних информационных объектов с помощью множества имен, разделенных пробелами;
- **NMTOKEN** – имя XML;
- **NOTATION** – имя в обозначении, объявленном в DTD;
- **NMTOKENS** – множество имен XML, разделенных пробелами.

Например, предположим, что в документе содержится тег вида:

```
<greeting language="Russia">
```

Приведенный тег может быть объявлен в DTD следующим образом:

```
<!ELEMENT greeting #PCDATA>
<!ATTLIST greeting language CDATA "English">
```

Здесь в первой строке указывается, что элемент **greeting** может содержать контролируемые символьные данные; во второй строке для этого элемента задается атрибут **language** типа CDATA и для атрибута устанавливается значение "English", используемое по умолчанию.

Комментарии в XML, как и в HTML, задаются между открывающим <!-- и закрывающим --> тегами.

Информационные объекты

Обычно документ XML состоит из двух частей: пролога и данных. В свою очередь, пролог включает объявление XML и описание типа документа DTD, данные представляют собой размеченный текст. В общем случае документ XML может содержать данные и объявления из различных источников: баз данных, сценариев и других источников данных. Таким образом, можно сказать, что документ XML состоит из пролога, за которым следуют информационные объекты (**entity** – сущности или объекты). В качестве информационного объекта может быть файл, запись базы данных или другой элемент с данными.

Информационные объекты позволяют задавать имена для наборов символьных данных или файлов, так чтобы их можно было включать в состав

XML-документа. Например, если в документе многократно используется символьная строка с информацией об издательстве, то целесообразно создать информационный объект с помощью следующего кода:

```
<!ENTITY redBHV "СПб.: ВНВ — Санкт-Петербург">
```

Теперь в документе можно указать ссылку вида

```
&redBHV
```

При обработке документа синтаксическим анализатором каждая такая ссылка будет заменена строкой

```
"СПб.: ВНВ — Санкт-Петербург"
```

Приведенный нами пример показывает использование *внутреннего* информационного объекта, описание которого содержится в самом документе. Кроме того, допустимо использование *внешних* информационных объектов, которые включают имя и ссылку на контейнер, хранящий его содержание. Объявление внешних информационных объектов выполняется с помощью кода следующего формата:

```
<!ENTITY name (PUBLIC|SYSTEM) "content">
```

Здесь строка name задает имя информационного объекта, ключевые слова PUBLIC и SYSTEM указывают на доступность информационного объекта в документах.

В XML имеется 5 *предопределенных* общих ссылок: <, >, ', " и &, которые ссылаются на текстовые информационные объекты (символы): <, >, ' и " соответственно. Они используются в документах XML, чтобы избежать неправильной интерпретации соответствующих символов как части разметки документа. К примеру, символы < и > при их непосредственном использовании могут интерпретироваться как начальный и конечный теги.

Определение типа документа

Создание отдельного определения типа документа DTD (Data Type Definition) для каждого документа XML обеспечивает следующее:

- возможность аналитическому анализатору проверять корректность документа;
- удобство задания и изучения структуры документа определенного типа;
- возможность применять не только элементы и атрибуты, но и информационные объекты.

В составе DTD имеются пять частей:

- типы документов;
- элементы;

- атрибуты;
- информационные объекты;
- комментарии.

Определение типа документа может размещаться внутри документа XML, в этом случае имеем *внутреннее DTD*, или задано во внешнем файле — *внешнее DTD*. Применительно к внешним DTD различают системное (SYSTEM) и общее (PUBLIC) определения типа документа. Системное DTD может использоваться одним или несколькими документами XML, общее DTD может использоваться в любом документе XML.

Одним из важнейших назначений DTD является определение структуры документа как иерархии составляющих документ элементов. В частности, применительно к книге соответствующий внутренний DTD и сам документ XML могут быть заданы следующим образом.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ENTITY % phras "PCDATA | emphasis">
<!ELEMENT book (title_page, content)>
<!ELEMENT title_page (author, title)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (part, chapter+)>
<!ELEMENT part (#PCDATA)>
<!ELEMENT chapter (#PCDATA)>
]>
<book>
<title_page>
<author> Ivanov Dm. </author>
<title> Name of book </title>
</title_page>
<content>
<part> Name of Part </part>
<chapter> Text of Chapter1 </chapter>
<chapter> Text of Chapter2 </chapter>
<chapter> Text of Chapter3 </chapter>
</content>
</book>
```

Здесь первая строка кода указывает, что это XML-документ, и определяет используемую систему кодирования — 8-битовую кодировку Unicode, соответствующую символам ASCII. Тег `<!DOCTYPE book [` указывает на начало

DTD — описания типа документа с именем book, а тег]> указывает на окончание DTD. В строке

```
<!ELEMENT content (part, chapter+)>
```

знак “+” указывает на то, что элементов chapter (глава) в родительском элементе content (содержание) книги может быть несколько.

При обнаружении ошибок в исходном коде документа XML обозреватель выдает сообщение о характере и местоположении первой из них, как показано на рис. 14.5. Отметим, что при подготовке документов HTML возможность контроля синтаксических ошибок просто отсутствует.

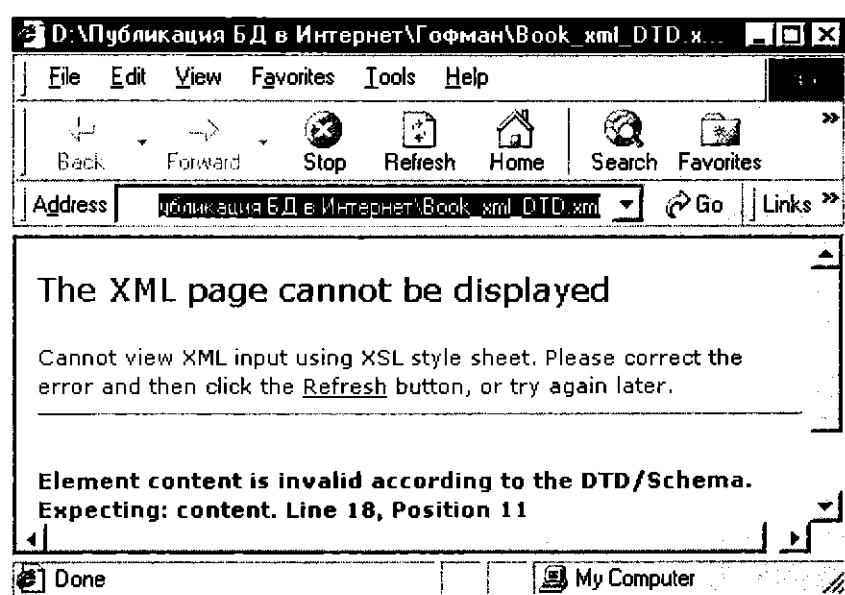


Рис. 14.5. Сообщение об ошибке в окне обозревателя

Для приведенного документа XML и его описания DTD в окне обозревателя Microsoft Internet Explorer 5.0 будет отображаться следующая структура (рис. 14.6):

Для создания *внешнего* DTD следует поместить объявления всех размещаемых в нем элементов, атрибутов и информационных объектов в отдельном файле с расширением DTD, например book.dtd. В состав внешнего DTD нужно поместить также инструкцию по обработке кода XML, включив строку

```
<?xml version="1.0" encoding="UTF-8" ?>
```

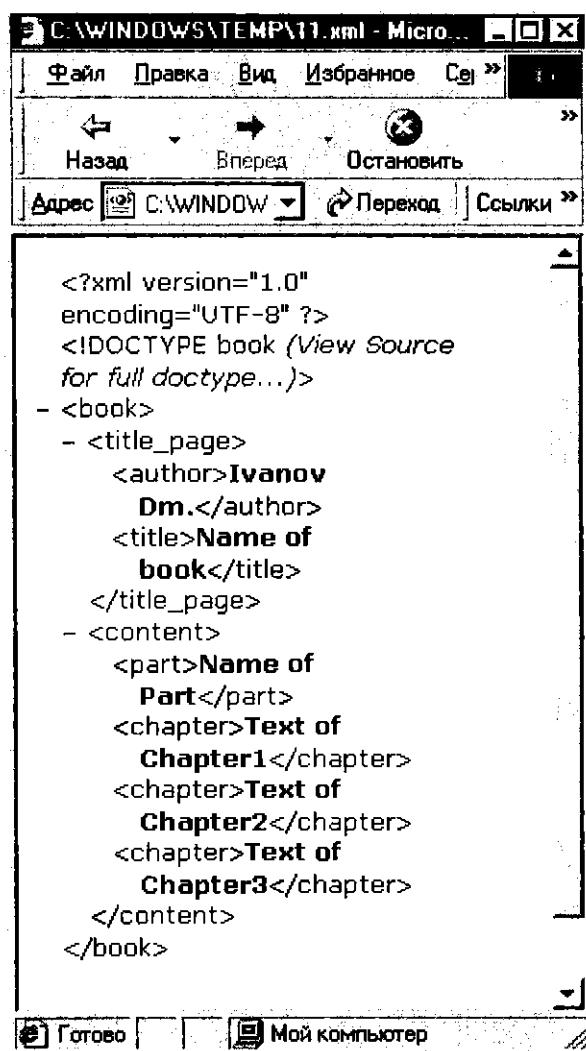


Рис. 14.6. Вид структуры в окне обозревателя

Чтобы получить доступ к внешнему DTD, размещенному в сети, в XML-документ нужно включить следующую строку

```
<!DOCTYPE book SYSTEM "http://www.xom.com/dtds/book.dtd">
```

Здесь "http://www.xom.com/dtds/book.dtd" — адрес размещения файла DTD в сети. Если DTD является общим, в приведенной строке вместо слова SYSTEM требуется указать PUBLIC.

Стилевое оформление XML-документа

Для стилевого оформления XML-документов служит расширяемый язык стиля XSL (eXtensible Style Language). Этот язык специально предназначен для стилевого форматирования XML-документов. По своим возможностям и целевому назначению язык XSL занимает промежуточное положение между каскадными таблицами стилей CSS (Cascading Style Sheets) и языком описания и семантики стиля документов DSSSL (Document Style Semantics and Specification Language). Последние два средства были созданы для работы с HTML- и SGML-документами соответственно.

Механизм стилей позволяет создать для одного XML-документа несколько таблиц стилей, обеспечивающих разные сценарии отображения этого документа в зависимости от ситуации. К примеру, один и тот же документ требуется просматривать в обозревателе и распечатывать в виде бумажной копии для работы.

По существу XSL-документ одновременно является XML-документом и для проверки его правильности можно использовать те же анализаторы XML-документов. Как и вся технология XML, язык XSL находится в стадии активного развития и уточнения.

Основным вариантом практического применения XSL является преобразование XML-документов в формат HTML, хотя существуют и другие возможные способы его применения. При таком преобразовании на вход процессора таблицы стилей (процессора XSL) поступает XML-документ и соответствующая таблица стилей XSL этого документа. На первом этапе преобразования получается документ XML в новом формате, на втором этапе выполняется форматирование и отображение этого документа в формате HTML.

В частности, для того, чтобы выполнить преобразование документа XML в HTML-документ с использованием таблицы стилей XSL при помощи XSL-процессора MSXSL фирмы Microsoft, требуется задать соответствующую командную строку в окне DOS. Например:

```
C:\XML>msxml -i test.xml -s test.xsl -o res.html
```

В приведенной командной строке выполняется вызов программы msxml как XSL-процессора с подачей на вход XML-документа test.xml и таблицы стилей test.xsl, в качестве выходного HTML-документа указывается файл res.html.

Здесь ключи -s и -i указываются обязательно в том случае, если имя таблицы стилей и выходного документа не совпадают с именем входного XML-документа. Поэтому приведенная командная строка может быть записана в виде

```
C:\XML>msxml -i test.xml -o res.html
```

Применительно к таблице стилей XSL элемент документа определяется с помощью пары тегов <XSL>...</XSL>. Каждая таблица стилей

заключается в эти теги. Состав элемента XSL, описывающего таблицу стилей, могут образовывать элементы, определяемые с помощью следующих тегов:

- <IMPORT> – импорт таблиц стилей в XSL-документ;
- <DEFINE-MACRO> – формирование исполняемого макроса;
- <DEFINE-SCRIPT> – определение исполняемого в документе сценария;
- <ID> – связь исходного элемента с идентификационной меткой;
- <CLASS> – связь исходного элемента с классом.

При необходимости можно импортировать в текущий XSL-документ таблицу стилей из другого XSL-атрибута. Для этого в начале XSL-документа нужно поместить следующий код:

```
<XSL>
    <IMPORT HREF="путь_к_таблице_стилей/doc_styles.xsl"/>
    ...
</XSL>
```

Здесь выполнено подключение таблицы стилей из файла doc_styles.xsl, предполагается, что остальная часть документа описывает правила стилей для элементов в данном документе.

Для подключения таблицы стилей к XML-документу нужно записать строку кода, содержащую инструкцию **XML-STYLESHEET**, следующего вида:

```
<?XML-STYLESHEET HREF="путь_к_таблице_стилей/doc_styles.xsl
TYPE="TEXT/XSL" ?>
```

Здесь параметр **TYPE="TEXT/XSL"** указывает на то, что подключается XSL-документ.

Документы XML, использующие стили, содержат несколько *объектов потока*. Последние представляют собой структуры, которые описывают отдельные части документа в виде комбинации определенных в нем элементов, а также задают стили, определенные в связанном XSL-документе. Создание объектов потока выполняется с помощью *правил построения* путем выбора из объектов потока, предопределенных заранее.

В качестве объектов потока XSL могут быть использованы объекты DSSL и HTML. К примеру, к числу основных объектов потока DSSL, применяемых в XSL, относятся такие объекты, как:

- **paragraph** – абзац;
- **character** – символ;
- **box** – граница вокруг элемента;
- **link** – гиперссылка;
- **score** – линия подчеркивания или перечеркивания текста;
- **table** – таблица;

- **table-cell** – ячейка таблицы;
- **table-column** – столбец таблицы.

К числу элементов объектной модели HTML-документа, которые могут быть использованы в качестве объектов потока XSL, относятся такие, как: A, AREA, BASE, BODY, CAPTION, COL, COLGROUP, DIV, FORM, FRAMESET, SCRIPT, TD, TEXTAREA, TITLE, TR.

Структура форматирования элементов в выходном документе на основе элементов в исходном XML-документе определяется с помощью *правил построения*, имеющих следующий формат записи:

```
<rule>
  <шаблон>
    <действие>
  </rule>
```

Например, правило построения:

```
<rule>
  <target-element type="title">
    <P margin-left="1.2in" font="16pt "Sans Serif"">
  </P>
</rule>
```

задает форматирование элементов **<title>** в виде элементов **<P>** в HTML-документе, которые отображаются шрифтом гарнитуры Sans Serif, размером 16 пунктов и с левым полем в 1.2 дюйма.

Шаблоны служат для описания элементов исходного документа или элементов, на которые действуют правила построения, и имеют следующий формат:

```
<rule>
  <target-element type="элемент">
    <действие>
  </rule>
```

Например:

```
<rule>
  <target-element type="book">
    <P font-size="18pt">
  </P>
</rule>
```

В приведенном шаблоне значение параметра **type** определяет составную часть исходного XML-документа (**target-element** – целевой элемент), к которому будет применено приведенное далее действие (форматирование с заданным размером шрифта).

Для сравнения элементов с целью определения форматируемых элементов в шаблонах используются следующие символы подстановки (трафаретные символы — *wildcards*):

- <target-element/> — шаблон должен действовать на все элементы. Для распространения действия шаблона на дочерние элементы можно задать дополнительный отбор с помощью элементов <element>;
- <any>...</any> — шаблон действует на произвольное число элементов, вложенных в целевой элемент (<target-element/>).

Например, шаблон вида

```
<rule>
    <element type="title">
        <any>
            <target-element type="paragraph"/>
        </any>
    </element>
    <действие>
</rule>
```

задает указанное форматирование (<действие>) для всех элементов элементов *paragraph*, размещенных на любом уровне вложенности элементов *title*.

Для определения, какую часть шаблона (правило построения) нужно использовать при форматировании анализируемой части документа, в шаблонах используются два вида спецификаторов (специальных тестов), которые обеспечивают сравнение с содержимым элемента документа при невозможности задания четких критериев сравнения.

Спецификатор *has-value* определяет, имеет ли заданный атрибут произвольное значение или нет. В составе правила построения указанный спецификатор записывается в следующем формате:

```
<rule>
    <target-element type="element">
        <attribute="имя" has-value="yes">
        <действие>
    </rule>
```

При задании такого спецификатора для анализируемой части XML-документа будет применено форматирование, определяемое указанным в шаблоне действием, если целевой элемент документа (параметр *type*) имеет атрибут с любым значением.

Спецификатор *position* определяет позицию элемента в составе XML-документа и может принимать следующие четыре значения:

- *first-of-any* — первый из родственных элементов указанного типа;
- *last-of-any* — последний из родственных элементов указанного типа;

- **first-of-type** — первый из элементов указанного типа;
- **last-of-type** — последний из элементов указанного типа.

Используя указанный спецификатор, можно задать различное форматирование для однотипных (*of-type*) и родственного типа (*of-any*) элементов документа, различающихся своим положением (позицией) в документе. В частности, такими элементами могут быть главы в книге или фамилии авторов в списке. Например, пусть в шаблоне для форматирования абзацев заданы следующие два правила построения.

```
<rule>
    <target-element type="paragraph">
        <действие1>
    </rule>
<rule>
    <target-element type="paragraph" position="first-of-type">
        <действие2>
    </rule>
```

Здесь для первого абзаца (*position="first-of-type"*) мы задаем форматирование (*<действие2>*), отличное от форматирования (*<действие1>*) остальных абзацев.

В шаблоне могут указываться произвольные комбинации критериев выбора, задаваемых с помощью символов подстановки, атрибутов и спецификаторов.

Контрольные вопросы и задания

1. Назовите примеры прикладных задач публикации баз данных в Интернете.
2. Изобразите схему взаимодействия Web-обозревателя и Web-сервера с использованием программ расширения.
3. Дайте общую характеристику языкам подготовки сценариев.
4. Что представляют собой элементы управления ActiveX?
5. Каково назначение апплетов и сервлетов Java?
6. Дайте общую характеристику интерфейсу CGI.
7. Каковы назначение и основные характеристики интерфейсов ISAPI/NSAPI?
8. Что представляют собой ASP, PHP и IDC/HTX-страницы?
9. В чем отличие интерфейсов OLE DB, ADO и ODBC?
10. В каких случаях целесообразно применять статическую публикацию баз данных и в каких динамическую?
11. Охарактеризуйте протоколы, используемые для передачи гипертекста.
12. Приведите примеры структурных тегов языка HTML.
13. Какие теги языка HTML используются для создания форм?

14. Приведите примеры графических тегов и тегов задания гиперссылок.
15. Дайте общую характеристику языка XML.
16. Перечислите составляющие XML-документа.
17. Каково назначение определения типа XML-документа и как оно задается?
18. Каким образом задается стилевое оформление XML-документа?
19. Запишите XML-документ, описывающий книгу.
20. Приведите пример определения типа XML-документа, описывающего каталог книг.

Литература

1. *Вебер Дж.* Технология JavaФ в подлиннике / Пер. с англ. — СПб.: ВНВ—Санкт-Петербург, 1999.
2. *Вейнер П.* Языки программирования Java и JavaScript / Пер. с англ. — ЛОРИ, 1998.
3. *Дунаев С. Б.* Технологии Интернет-программирования. — СПб.: БХВ—Петербург, 2001.
4. *Коннолли Т., Бэгг К.* Базы данных. Проектирование. Реализация и сопровождение. Теория и практика. — М.: Вильямс, 2003.
5. *Матросов А. В., Сергеев А. О., Чайкин М. П.* HTML 4.0. — СПб.: БХВ-Санкт-Петербург, 2000.
6. *Мещеряков Е. В., Хомоненко А. Д.* Публикация баз данных в Интернете. — СПб.: БХВ-Петербург, 2001.
7. *Питс-Моултис Н., Кирк Ч.* XML / Пер. с англ. — СПб.: ВНВ—Санкт-Петербург, 1999.
8. *Уильямс Э. Э Барбер К., Ньюкирк П.* Active Server Page / Пер. с англ. -- СПб.: БХВ-Петербург, 2001.
9. *Фролов А. В., Фролов Г. В.* Базы данных в Интернете: практическое руководство по созданию Web-приложений с базами данных. — М.: Издательско-торговый дом «Русская редакция», 2000.

15. Web-приложения и Web-серверы

Современные Web-приложения применяются для создания информационных систем в сетях Интернет/интранет и обычно строятся как многоуровневые (многоуровневые) приложения, публикующие БД.

Введение многоуровневой архитектуры при большом числе клиентских компьютеров обусловлено необходимостью уменьшения нагрузки на сервер баз данных и линии связи. Построение Web-приложений, предназначенных для использования в интранете, может существенно отличаться от построения сетевых приложений БД на основе клиент-серверных технологий.

В этой главе описываются принципы функционирования и архитектура Web-приложений, особенности Web-приложений, публикующих БД в сетях Интернет/интранет, а также рассматриваются характеристика и особенности.

15.1. Принципы функционирования Web-приложений

Программные средства сетей Интернет/интранет включают новую категорию программ — Web-приложения. К Web-приложениям относят набор Web-страниц, сценариев и других программных средств, расположенных на одном или нескольких компьютерах (клиентских и серверных) и объединенных для выполнения прикладной задачи. Web-приложения, публикующие БД в Интернет, представляют отдельный класс Web-приложений.

Современные информационные системы в сетях Интернет/интранет, построенные на основе Web-приложений, использующих БД, по своей архитектуре и организации работы опираются на принципы, заложенные в многоуровневых клиент-серверных приложениях и принципах функционирования Интернета. Основы архитектуры многоуровневых клиент-серверных приложений рассмотрены в предыдущей главе. Web-приложения имеют ряд особенностей функционирования, заключающиеся в принципах работы Интернета. Кратко напомним основные из этих принципов.

Web-приложения выполняются на стороне Web-сервера, который находится на Web-узлах сети Интернет. Web-сервер обрабатывает запросы обозревателя на получение Web-страниц и отсылает требуемые данные обозревателю в формате Web-документов.

Обмен данными в сети Интернет осуществляется на аппаратном уровне на основе протокола TCP/IP и протокола более высокого логического уровня HTTP. Упрощенная схема функционирования Web-приложения приведена на рис. 15.1.

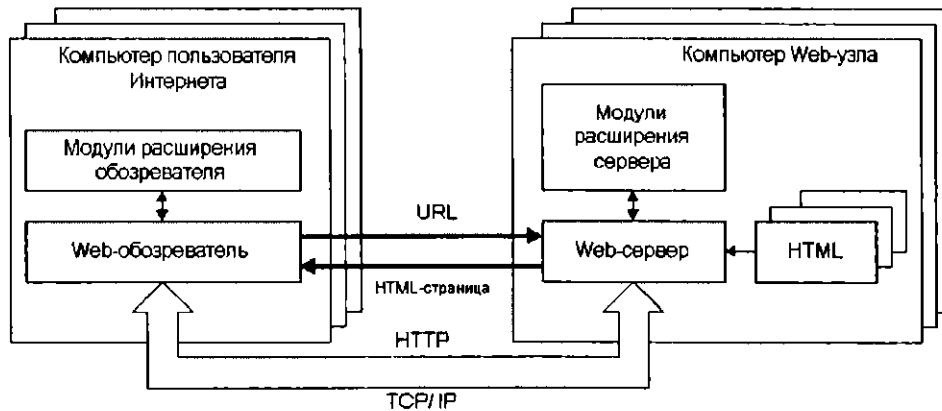


Рис. 15.1. Упрощенная схема функционирования Web-приложения

Напомним, что под Web-документом (Web-страницей) понимают документы, используемые в сети Интернет в форматах HTML, XML, шаблоны в форматах ASP, HTX и т. д.

Для доступа к Web-страницам используются специальные клиентские программы — обозреватели Web, находящиеся на компьютерах пользователей Интернета. Обозреватель формирует запрос на получение требуемой страницы или другого ресурса с помощью адреса URL. Функции обозревателя заключаются в отображении Web-страниц, генерированных сервером или модулями расширения, и отправке запросов пользователю Web-приложению. Обозреватель является связующим звеном между пользователем и Web-приложением. При этом Web-обозреватель устанавливает соединение с требуемым Web-узлом, используя различные протоколы передачи данных, нами рассматривается использование протокола HTTP.

Web-приложения в сетях интранет

Развитие технологий в сети Интернет привело к появлению новых архитектурных и технологических решений для локальных корпоративных интранет-сетей. Сети интранет построены на том же аппаратно-программном обеспечении, принципах и протоколах, что и сеть Интернет. В общем случае под сетью *интранет* понимают выделенную часть сети Интернет, в которой выполняется Web-приложение (информационная система).

Применение Интернет-технологий в корпоративных интранет-сетях позволяет повышать эффективность функционирования сетей и используемых в них информационных систем.

Приложения, построенные на основе использования Интернет-технологий, характеризуются надежностью и масштабируемостью, открытостью архитектуры, простотой изучения и использования.

Надежность WEB-приложений обусловлена устойчивостью работы программно-аппаратных средств сети Интернет, устойчивость к сбоям которых испытана в течение многих лет. Например, наиболее популярные WEB-сервера осуществляют обработку более 50 миллионов обращений в день, не имея при этом катастрофических сбоев.

Масштабируемость Web-приложений сетей инTRANет обеспечивается использованием многоуровневой архитектуры, позволяющей одно и тоже Web-приложение практически без переконфигурирования выполнять для инTRANет-приложений с различной архитектурой.

Открытость архитектуры инTRANет-приложений основывается на гибкой многоуровневой архитектуре и стандартизованных протоколах и форматах документов, доступных для модификации.

Простота изучения и использования инTRANет-приложений обусловлена стандартизацией пользовательского интерфейса на основе применения однотипного клиентского приложения — обозревателя со стандартным пользовательским графическим интерфейсом. Достаточно освоить принципы работы одной программы обозревателя, чтобы можно было работать с любыми инTRANет-приложениями.

Кроме того, использование инTRANет-приложений характеризуется значительным снижением денежных затрат (в десятки и сотни раз) на обслуживание, модернизацию и наращивание сети инTRANет по сравнению с традиционными корпоративными сетями, построенными на клиент/серверных технологиях. Важным достоинством сетей инTRANет является возможность развертывания на существующей инфраструктуре корпоративных локальных и глобальных сетей. Для построения сети инTRANет допускается простое встраивание в существующие корпоративные сети с использованием существующего аппаратного обеспечения.

При использовании Web-приложений в сети инTRANет может использоваться архитектура, показанная на рис. 15.2. Сеть инTRANет в общем случае имеет различную внутреннюю структуру, построенную на принципах Интернет. Причем сеть инTRANет может и не иметь выход в Интернет.

Сеть инTRANет строится на основе архитектуры распределенных приложений БД и архитектуры Web-приложений. Напомним, что взаимодействие между распределенными компонентами такой сети осуществляется на аппаратном уровне на базе протокола TCP/IP, а на логическом уровне — на принципах, заложенных в протоколе HTTP.

В архитектуре инTRANет-сети сегменты могут иметь развитую структуру, обеспечивающую разграничение доступа и конфиденциальность информации. Такая структура реализуется с помощью маршрутизаторов (устройств-коммутаторов, используемых для поиска необходимого узла сети), распределенных в пределах группы клиентов сети, либо путем использования центрального маршрутизатора и многочисленных коммутаторов.

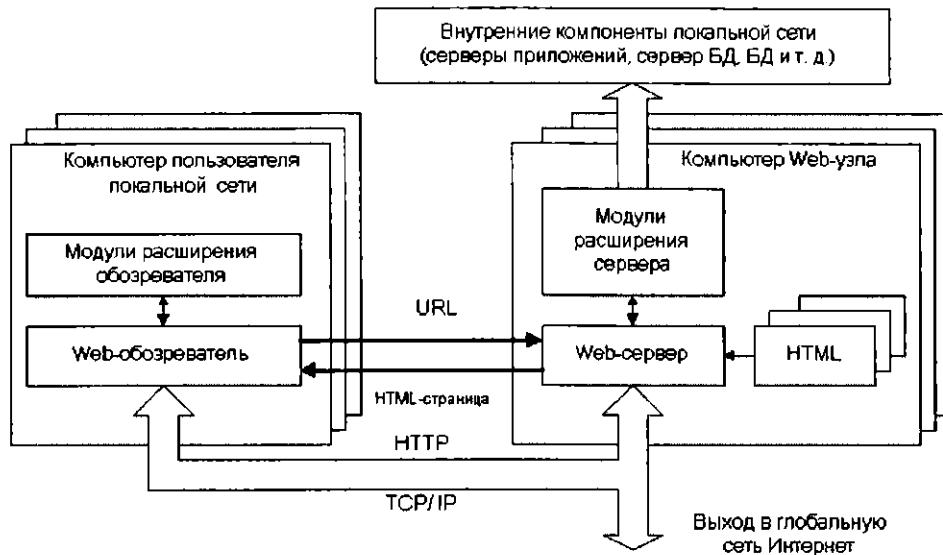


Рис. 15.2. Схема функционирования Web-приложения с использованием модулей расширения

В качестве клиентских приложений в этой архитектуре выступают Web-обозреватели, которые обращаются с запросами к серверу БД или к серверу приложений через Web-сервер. В зависимости от используемой архитектуры Web-сервер может находиться на сервере БД или на сервере приложений.

В функции Web-сервера в сети интранет входит обработка запросов Web-обозревателей на получение информации из разделяемых БД, преобразование этих запросов (может выполняться модулями расширения Web-сервера) в SQL-запросы или другие форматы, понятные для сервера БД или сервера приложений.

Интранет-приложение предоставляет следующие дополнительные возможности:

- Реализация концепций удаленного доступа и управления. Концепция *удаленного доступа* подразумевает возможность удаленного подключения к интранет-сети, то есть подключение к сети интранет из любого компьютера сети Интернет. Под *удаленным управлением* понимается подключение к локальной сети и выполнение функциональных операций по управлению ресурсами локальной сети с удаленного компьютера. Для реализации дистанционного управления необходимо наличие специального сервера удаленного доступа и специального программного обеспечения на удаленном компьютере;

- Обеспечение доступа в Интернет клиентов сети. При этом становятся доступными услуги, предоставляемые сетью Интернет, связанные с возможностью получения свежей информации в различных сферах, становятся доступны услуги электронной почты, возможность обмена информацией с внешними информационными и деловыми источниками, а также использование приложений, находящихся в Интернете, использование Интернета для рекламы и т. д.

Простота создания и наращивания сети интранет позволяет быстро создавать локальные, защищенные сетевые системы с технологией клиент-сервер, доступные для быстрого освоения. Сеть интранет может быть построена на основе использования Web-сервера в локальной сети или на основе услуг, предоставляемых внешним Web-сервером, находящимся в сети Интернет. Такие услуги, как правило, предоставляет провайдер, обеспечивающий возможность использования функций своего Web-сервера. Многие компании используют совмещенную структуру Web-серверов. При этом сама локальная сеть строится с использованием собственного Web-сервера, а выход в глобальную сеть осуществляется через Web-сервер провайдера, публикующий маркетинговую информацию компании.

Современные сети интранет имеют различное назначение и особенности конкретной реализации. При этом выделяют следующие принципы построения сетей интранет:

- иерархическая архитектура интранет-приложений. При такой архитектуре информация размещается иерархически сверху вниз. Для этого создают узловые серверы, на которых размещается наиболее важная информация, используемая совместно несколькими отделами или подсетями. При этом серверы могут составлять иерархическую структуру. На верхнем уровне находится сервер с информацией, необходимой для клиентов всей сети, а на нижних уровнях иерархии находятся специализированные серверы, содержащие информацию, используемую для групп клиентов сети. Таким образом обеспечивается максимально быстрый доступ к любой внутрисетевой информации и снижаются затраты на обновление корпоративной информации;
- использование «двусторонней обратной связи» с клиентами сети для обработки их запросов. При этом процесс восстановления при сбоях или перерывах в работе происходит с использованием механизма транзакций. Интранет-приложения, построенные на основе этого принципа, называют транзакционными Web-приложениями. Их целесообразно использовать, к примеру, при создании Интернет-магазинов с помощью промышленных СУБД. При этом можно эффективно отслеживать осуществление всех операций цикла продажи товара (заказа товара, оплаты и доставки товара);

- использование группового способа общения. В этом случае объединяются группы новостей с возможностью прямого обмена информацией между различными членами группы клиентов и разграничение доступа к информации для пользователей вне группы.

Применение архитектуры Web-приложений в сетях интранет по сравнению с традиционными архитектурами локальных сетей имеет следующие преимущества:

- стандартизация пользовательского интерфейса — использование обозревателя в качестве универсальной клиентской программы позволяет упростить процесс обучения пользователей и обслуживания клиентских компьютеров;
- более удобное администрирование и конфигурирование заключается в том, что в сети интранет вносимые в серверах приложений и серверах БД изменения не затрагивают клиентский уровень, то есть не надо вносить изменения в огромное количество компьютеров пользователей сети при изменении конфигурации БД (достаточно изменить текст сценария, хранящийся на Web-сервере);
- удешевление установки и лицензирования клиентских компьютеров пользователей сети интранет.

Для расширения возможностей клиентской части (обозревателя) и серверной части разрабатывают модули расширения обозревателя и сервера, используемые для динамического управления интерфейсными объектами (компонентами) Web-документа.

Последовательно рассмотрим взаимодействие Web-приложения с дополнительными компонентами: модулем расширения клиентской части и модулем расширения серверной части (рис. 15.1).

Web-приложения с модулями расширения сервера

Архитектура Web-приложений с модулями расширения сервера (рис. 15.3) может включать стандартные модули расширения — dll-библиотеки, реализующие, например, технологии ASP, IDC/HTX, объекты ActiveX. Кроме того, могут подключаться дополнительные модули, разработанные с использованием интерфейсов CGI, ISAPI и др.

В этом случае в функции Web-сервера входят обработка запросов Web-обозревателей пользователей сети, вызов (загрузка) соответствующего модуля расширения сервера и передача ему параметров запроса. В результате обработки запроса модулями расширения сервера формируется Web-документ с использованием различных HTML-шаблонов. Готовый HTML-документ отсылается Web-обозревателю в формате протокола HTTP.

Web-сервер формирует динамически Web-страницы различными способами и отсылает готовые Web-страницы в формате протокола HTTP.

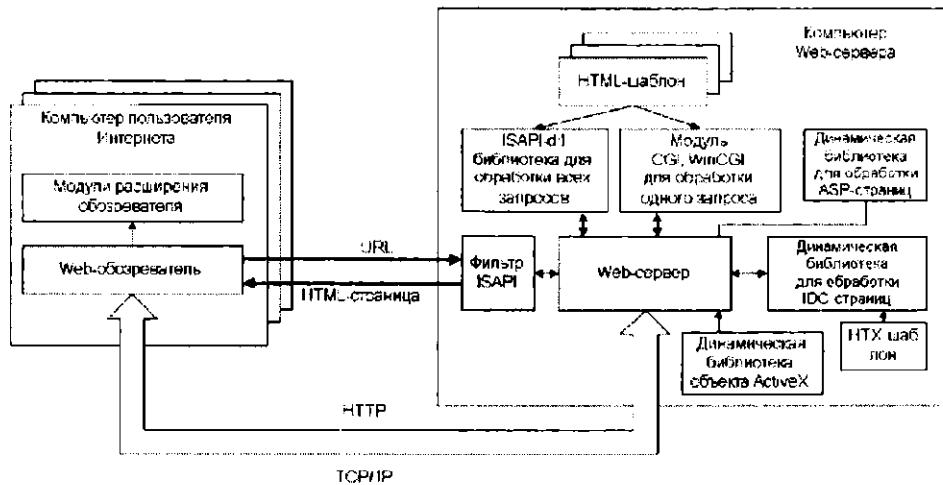


Рис. 15.3. Архитектура Web-приложения с модулями расширения сервера

Различают пассивное и активное состояние Web-сервера. Так, Web-сервер находится в *пассивном* состоянии, если формируемый им документ содержит статическую информацию, то есть на Web-странице отсутствуют средства ввода и обработки запросов к серверу.

В *активном* состоянии Web-сервер находится при динамическом создании Web-документов в ответ на запрос пользователя (рис. 15.3) или в случае, когда в обозреватель загружены различные интерактивные элементы формы. Для публикации БД основной интерес представляет активный Web-сервер, реализуемый с помощью модулей расширения Web-сервера.

Для организации связи программных расширений Web-сервера с БД используются также современные интерфейсы доступа к данным OLE DB, ADO и ODBC. Эти интерфейсы являются промежуточным уровнем между источником данных и приложением, в качестве которого выступают программные расширения Web-сервера.

Для создания модулей расширения Web-сервера могут использоваться интерфейсы CGI, WinCGI или интерфейсы программирования API.

Интерфейс CGI является стандартным протоколом взаимодействия между Web-сервером и модулями расширения, которые могут применяться для выполнения дополнительных функций, не поддерживаемых сервером. Например, такие модули используются для обработки получаемой от пользователя информации, динамического формирования Web-документа, публикации БД на Web-странице и т. д.

Напомним, что интерфейсу CGI соответствуют обычные консольные приложения операционной системы DOS. Обмен информацией между сервером

и модулем расширения осуществляется с помощью стандартного потокового ввода/вывода, передача управляющих параметров может организовываться через переменные окружения операционной системы или через параметры URL-адреса модуля расширения.

WinCGI протокол отличается от протокола CGI только тем, что управляющие параметры передаются через INI-файл, а входной и выходной потоки данных перенаправлены в специальные файлы.

Для написания CGI-программ подходит практически любой язык программирования, обеспечивающий доступ к переменным среды и ввод-вывод через стандартные потоки STDIN и STDOUT. Для написания CGI-программ подходят среды разработки C++Builder, Delphi, Visual C++ и Java. Кроме того, для этих целей можно использовать интерпретаторы языков Perl, PHP, предназначенные для различных интерфейсов Web-серверов.

Для запуска CGI-модуля обозреватель должен сформировать запрос к серверу с указанием адреса URL этого модуля.

Для запуска модуля расширения можно использовать следующие способы:

- задание адреса URL модуля CGI в строке адреса обозревателя;
- посылка обозревателем серверу запроса на выполнение CGI-модуля при выборе пользователем ссылки, атрибут «`href`» которой содержит адрес этого модуля;
- нажатие кнопки типа «`Submit`», входящей в форму, у которой атрибут «`Action`» содержит адрес URL CGI-модуля.

После передачи запроса обозревателя CGI-приложению сервер передает ему также данные из командной строки запроса. CGI-приложение формирует ответ и помещает его в выходной поток (на стандартном устройстве вывода), затем сервер посыпает этот ответ с использованием протокола HTTP обратно обозревателю. В случае параллельной обработки нескольких запросов сервер запускает отдельный процесс для обработки каждого запроса. Причем для каждого процесса создается копия модуля расширения в памяти компьютера, на котором находится Web-сервер. Поэтому недостатками этого протокола является невысокая скорость обработки запросов и повышенная загрузка Web-сервера.

При большом размере исполняемого CGI-файла сильно увеличивается время отклика сервера на запрос обозревателя, поскольку потребуется время для загрузки модуля CGI с диска в память. Причем, если CGI-программа является интерпретируемой (например, написана на языке PHP или Perl), она будет выполняться еще медленнее, так как в этом случае, кроме загрузки модуля CGI, загружается интерпретатор PHP или Perl и производится интерпретация команд. При наличии сотен или тысяч обращений к серверу произойдет запуск сотен или тысяч CGI-программ соответственно, каждая из которых будет обрабатывать соответствующий запрос обозревателя.

Отметим, что в некоторых операционных системах, в частности Unix, могут повторно использоваться уже загруженные программные коды модуля CGI первого процесса, создавая для каждого нового обращения только свой экземпляр данных.

WinCGI протокол отличается от протокола CGI тем, что управляющие параметры передаются через INI-файл, а входной и выходной поток данных перенаправлены в специальные файлы. Этот протокол является реализацией протокола CGI для операционной системы Windows 3.1. Сервер передает данные CGI-программам через INI-файл Windows в формате «параметр-значение». Программа WinCGI читает этот файл и получает все данные, передаваемые ей из формы и автоматически генерируемые обозревателем.

INI-файл Windows состоит из нескольких специальных секций, в которых находятся различные параметры в виде текстовых строк. В системных секциях, которые автоматически генерируются обозревателем, находится информация о типе доступа, составных элементах URL-запроса, с помощью которого WinCGI-модуль был загружен. Там же находится информация для настройки работы WinCGI-модуля, информация о сервере, путь к файлу, в который помещаются данные, отсылаемые сервером клиенту после выполнения WinCGI-модуля, «дополнительные» параметры, которые включены в URL-запрос. В остальном функционирование этого интерфейса аналогично интерфейсу CGI.

Более перспективными интерфейсами для разработки дополнительных модулей расширения Web-сервера являются интерфейсы ISAPI/NSAPI. При использования этих интерфейсов модули расширения реализуются в виде библиотек DLL. Рассмотрим принципы функционирования модулей ISAPI.

Запуск модуля расширения выполняется сервером в ответ на первый запрос обозревателя (в случае CGI-интерфейса загрузка осуществляется каждый раз) на загрузку URL-адреса этого модуля. Загрузка модулей ISAPI осуществляется теми же способами, что и загрузка модулей CGI. Обмен информацией между сервером и модулем расширения осуществляется с помощью специальных объектов Request и Response. Сервер передает параметры запроса модулю расширения с помощью объекта Request и получает сформированный Web-документ с помощью объекта Response.

В многопользовательском режиме работы сервера при обработке сервером последующих запросов к модулю расширения сервер использует уже загруженный экземпляр динамической библиотеки. Такой механизм взаимодействия сервера и модуля расширения обеспечивает экономию ресурсов сервера и увеличение скорости обработки запросов.

Однако использование интерфейса API требует от разработчика модуля расширения соблюдения особых мер по обеспечению устойчивости этого модуля к сбоям. При возникновении катастрофического сбоя потребуется перезапуск Web-сервера.

Интерфейс ISAPI может применяться также для создания ISAPI-фильтров, которые, в отличие от модулей ISAPI, могут использоваться для контроля всего потока данных между сервером и обозревателем на уровне протокола HTTP. ISAPI-фильтры можно применять для динамической перекодировки, шифрования, сбора статистической информации о работе сервера.

Для уменьшения нагрузки на Web-сервер часть функций, связанных с предварительной обработкой запросов и введенных данных, целесообразно выполнять на стороне клиента (в Web-обозревателе). Эту задачу решают модули расширения клиентской части. В следующем разделе рассмотрим основные особенности архитектуры Web-приложений, использующих модули расширения обозревателя.

Web-приложения с модулями расширения клиентской части

В случае модулей расширения клиентской части (активность на стороне клиента) используют апплеты, подключаемые программы, ActiveX-объекты и сценарии. Эти технологии могут использоваться для создания динамических эффектов при просмотре Web-страницы. Архитектура Web-приложения с использованием модулей расширения клиентской части показана на рис. 15.4.

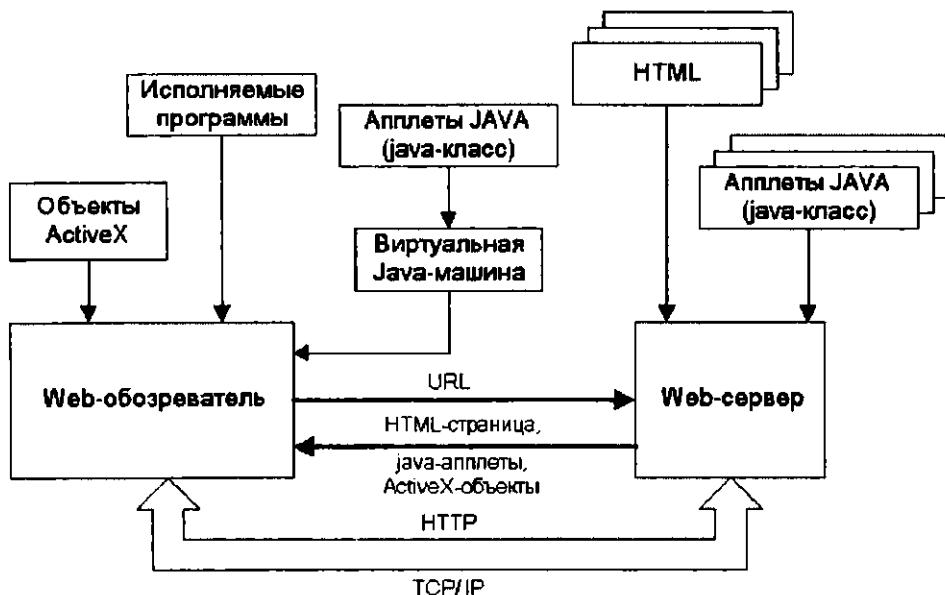


Рис. 15.4. Схема Web-приложения с использованием модулей расширения обозревателя

Элементы управления ActiveX представляют собой вид модулей расширения, которые могут использоваться на стороне клиента или на стороне сервера. Они реализуются с помощью динамических библиотек DLL и могут быть встроены в Web-документ как дополнительные интерфейсные элементы. Механизм работы элементов управления ActiveX позволяет из программного кода этих объектов получать неограниченный доступ к локальным ресурсам компьютера пользователя. Из элемента управления ActiveX имеется возможность передавать на сервер любую информацию с компьютера пользователя. С точки зрения обеспечения безопасности локальных данных пользователей использование элементов управления ActiveX не всегда оправдано.

Исполняемые программы являются первым поколением клиентских расширений. Для организации работы такой программы в Web-приложении необходимо предварительно устанавливать такую программу и конфигурировать обозреватель. Поэтому при использовании механизма исполняемых программ очень трудно обеспечить совместимость такого Web-приложения для различных обозревателей. Кроме того, использование такой технологии нарушает принцип стандартизации клиентского интерфейса.

Апплеты Java применяются для создания динамически формируемого интерфейса пользователя. Апплет представляет собой байтовый код, который интерпретируется виртуальной Java-машиной, входящей в состав обозревателя. У апплетов отсутствует возможность производить запись на диск пользователя. Поэтому использование механизма апплетов гарантирует целостность локальных данных пользователей.

Загрузка апплета производится при загрузке Web-документа. Принцип функционирования апплета включает возможность подключения Java-классов, хранящихся на сервере. Таким образом, сам апплет содержит только необходимый код, а при последующем взаимодействии пользователя с интерфейсными элементами апплета могут дополнительно подгружаться требуемые Java-классы, реализующие дополнительные функции.

Байтовый код апплета может начать выполняться сразу после загрузки в компьютер клиента или активизироваться с помощью специальных команд.

Используя библиотеку классов Java, можно создавать мультимедийные страницы и организовывать распределенные процедуры вычислений с использованием различных серверов и с использованием различных протоколов.

Используемый для выполнения апплетов интерпретатор байт-кода, входящий в состав виртуальной машины Java, имеетстроенную систему безопасности, блокирующую операции записи информации на диск и операции адресной арифметики.

Апплет может быть специализирован для работы с внешними базами данных. С этой целью в Java включены наборы классов для поддержки графического пользовательского интерфейса и классы для доступа к БД. С помощью

этих классов аплет может получить от пользователя информацию ввода параметров запроса к базе данных.

Для включения дополнительного действия в Web-приложение достаточно включить тег аплета в Web-документ и поместить аплет-класс в библиотеку аплетов на сервере. При этом изменения в конфигурацию Web-сервера вносить не нужно.

Для взаимодействия Java-апледта с внешним сервером баз данных разработан специализированный протокол JDBC (Java DataBase Connectivity – совместимость Java с базами данных), который построен на принципах интерфейса ODBC и применяется для стандартизации кода Java-апледта при организации доступа к различным СУБД. Протокол JDBC является посредником между Java-кодом и драйвером ODBC.

Сравним достоинства и недостатки использования технологии Java и наиболее распространенного в настоящее время интерфейса CGI. Использование технологии Java-апледтов реализует более гибкий механизм. Апледт выполняется локально на машине пользователя, поэтому он может обеспечивать динамическое взаимодействие с пользователем гораздо быстрее. Кроме того, апледт может использоваться для выполнения функций, не доступных CGI-модулю. Однако с точки зрения обеспечения безопасности данных компьютеров клиентов сети использование CGI-модуля более целесообразно, так как CGI-модуль выполняется на стороне сервера и получить доступ к ресурсам компьютера пользователя сети не может.

На этом мы закончим рассмотрение общих принципов построения Web-приложений и перейдем к рассмотрению архитектуры Web-приложений, публикующих БД.

15.2. Архитектура Web-приложений, публикующих БД

При публикации БД на Web-страницы в архитектуру Web-приложений вводятся дополнительные уровни, включающие сервер БД, сервер приложений и источник БД.

При такой архитектуре Web-сервер передает запрос на генерацию Web-страниц программе-расширению Web-сервера, которая формирует требуемый документ по информации из БД и затем Web-сервер отсылает готовые Web-страницы обратно обозревателю. Для формирования динамических страниц используются различные средства и технологии: ASP и IDC/HTX-страницы, программы-расширения сервера на основе интерфейсов CGI и ISAPI.

При использовании ASP, PHP и IDC/HTX-страниц запрос на получение динамически формируемой Web-страницы передается специальным динамическим библиотекам, входящим в состав Web-сервера. Например, если в ка-

честве Web-сервера используется Personal Web Server и публикация осуществляется средствами IDC/HTX, то применяется динамическая библиотека «`httpodbc.dll`». Такие библиотеки анализируют файл ASP или IDC и HTX-файлы, которые используются в качестве шаблонов.

Напомним, что при публикации БД на Web-страницах используются статическая и динамическая публикация Web-страниц, содержащих информацию из базы данных. При *статистической* публикации подготовка Web-страницы осуществляется с помощью обычных приложений БД, которые формируют Web-документ на жестком диске. Причем, Web-документы создаются и хранятся на Web-сервере до поступления запроса пользователя на их получение. Такой способ не позволяет публиковать БД в реальном масштабе времени, и, естественно, этот способ публикации используется в случае когда информация в базе данных обновляется относительно редко. Тем не менее при статистической публикации уменьшается нагрузка на сервер при обработке запросов, так как серверу в этом случае нет необходимости обращаться с запросом к серверу БД, а достаточно передать готовую Web-страницу обозревателю.

При *динамической* публикации Web-страницы создаются после поступления запроса пользователя на Web-сервер. Поступивший запрос на генерацию Web-страниц Web-сервер передает программе-расширению сервера, которая формирует требуемый документ, и затем Web-сервер отсылает готовые Web-страницы обратно обозревателю.

Программы-расширения Web-сервера, публикующие базы данных, используют принципы, на которых строятся приложения БД. При статистической и при динамической публикации должна использоваться архитектура приложений БД, дополненная характерными компонентами архитектуры Web-приложений. Рассмотрим архитектуру Web-приложений, использующих БД.

Двухуровневые Web-приложения

При двухуровневой архитектуре Web-приложений источник БД хранится на том же компьютере, где находится Web-сервер. Для доступа к источнику БД используются модули расширения. В простейшем случае в архитектуру Web-приложений добавляется источник БД (рис. 15.5).

Схема функционирования Web-приложения при такой архитектуре заключается в следующем. Обозреватель для начала работы с Web-приложением отсылает URL-адрес главной страницы Web-приложения Web-серверу. Последний, обработав URL запроса, высылает главную страницу Web-приложения обозревателю в формате HTML. Эта страница несет общую информацию о Web-приложении и позволяет выбрать требуемую для пользователя функцию из ряда других, предоставляемых этим Web-приложением. Далее возможны несколько вариантов работы Web-приложения.

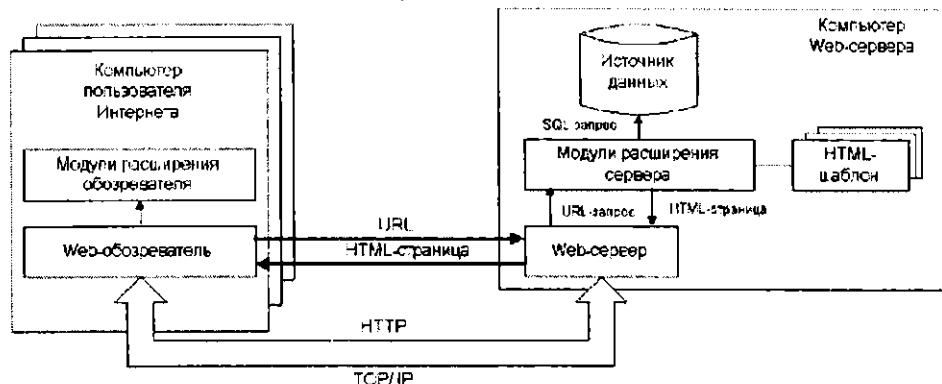


Рис. 15.5. Архитектура Web-приложения, использующего БД

Если пользователю нужна определенная информация из БД, то обозреватель по ссылке, находящейся в загруженной HTML-странице, формирует URL запрос к модулю расширения сервера, при этом могут использоваться различные технологии в зависимости от используемого Web-сервера на Web-узле и других особенностей работы Web-приложения.

Например, если на Web-узле установлен Web-сервер Microsoft Internet Information Server, то может использоваться технология ASP-страниц, IDC/HTX-страниц, CGI или ISAPI-технология, а в случае установки сервера Apache может применяться CGI-технология.

При необходимости формирования параметризованного URL на уровне обозревателя могут использоваться сценарии на JavaScript для проверки правильности ввода параметров запроса.

После того как пользователь выбрал требуемую ссылку, обозреватель отсылает URL Web-серверу. Для обработки такого запроса сервер вызывает требуемый модуль расширения и передает ему параметры URL. Модуль расширения сервера формирует SQL-запрос к БД.

Из модуля расширения сервера доступ к БД может осуществляться различными способами и на основе различных интерфейсов. Например, в случае использования технологии ASP-страниц применяется несколько уровней интерфейсов: объектная модель ADO, объектный интерфейс OLE DB, интерфейс ODBC. Также возможен вариант непосредственного доступа к БД. Например, в случае модуля ISAPI, разработанного в среде Delphi, для доступа к БД может использоваться один посредник — драйвер BDE (Borland DataBase Engine), входящий в состав программных средств модуля расширения сервера.

Недостатки рассмотренной двухуровневой архитектуры состоят в следующем:

- повышенная нагрузка на Web-сервер, заключающаяся в том, что вся работа по обработке URL-запросов, извлечению информации из БД и фор-

мированию HTML-страниц выполняется Web-сервером и модулями расширения Web-сервера;

- низкий уровень безопасности, связанный с невозможностью, например, обеспечить требуемый уровень защиты информации в БД от сбоев во время обращения к базе данных из модуля расширения сервера или конфиденциальности информации БД от администратора Web-узла.

Для преодоления указанных недостатков применяются Web-приложения с большим числом уровней. Перейдем к их рассмотрению.

Трехуровневые Web-приложения

При внесении в Web-приложение промежуточного уровня, основанного на технологии клиент-сервер, его архитектура расширяется до трехуровневой. При такой архитектуре клиентский уровень занимает обозреватель, на уровне сервера находится сервер БД, на промежуточном уровне находятся Web-сервер и модули расширения сервера. Модуль расширения сервера выступает преобразователем протоколов между клиент-серверным приложением БД и Web-сервером (рис. 15.6).

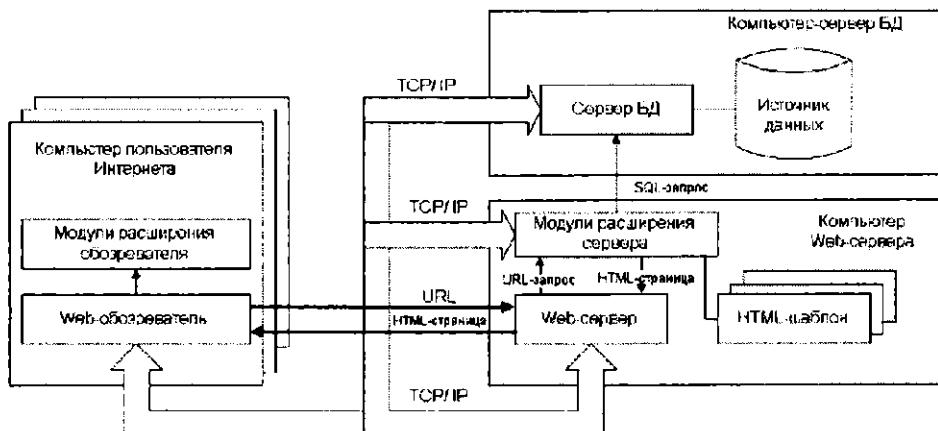


Рис. 15.6. Архитектура трехуровневого Web-приложения, использующего БД

Введение уровня Web-сервера в клиент-серверные приложения БД расширяет возможность их применения как для межплатформенного приложения. Принципы взаимосвязи обозревателя и Web-сервера остаются те же, что и в предыдущей архитектуре. Отличия этой архитектуры заключаются в организации взаимодействия модуля расширения сервера и источника данных.

Для получения данных модуль расширения Web-сервера формирует и отсылает SQL-запрос удаленному серверу БД (SQL-серверу). На компьютере,

где установлен удаленный сервер БД, содержится и база данных. SQL-сервер обеспечивает выполнение запроса и выдачу модулю расширения Web-сервера результатов запроса.

Таким образом, в трехуровневой архитектуре вся обработка SQL-запроса выполняется на удаленном сервере. Достоинства такой архитектуры по сравнению с предыдущей состоят в следующем:

- уменьшение сетевого трафика — в сети циркулирует минимальный объем информации;
- увеличение уровня безопасности информации, поскольку обработка запросов к базе данных выполняется сервером БД, который управляет доступом к базе данных, запрещая одновременное изменение одной записи различными пользователями и реализуя механизм транзакций;
- повышение устойчивости Web-приложения к сбоям;
- взаимозаменяемость компонентов архитектуры трехуровневого Web-приложения;
- снижение сложности модулей расширения Web-сервера, в которых отсутствует программный код, связанный с контролем БД и разграничением доступа к ней.

Недостатком рассмотренной архитектуры является увеличение времени обработки запросов, связанное с дополнительным обращением по сети к серверу БД. Для устранения этого недостатка между сервером БД и Web-сервером должны использоваться высокоскоростные надежные линии связи.

Многоуровневые Web-приложения

Дальнейшее развитие архитектуры Web-приложений и технологии «клиент-сервер» привело к появлению многоуровневой архитектуры, в которой между модулем расширения Web-сервера и базой данных, кроме сервера БД, дополнительно вводится сервер приложений. *Сервер приложений* является промежуточным уровнем, который обеспечивает организацию взаимодействия клиентов («тонких» клиентов) и сервера БД (рис. 15.7).

Напомним, что сервер приложений может использоваться для выполнения различных функций, которые в предыдущей архитектуре выполнялись сервером БД или модулем расширения Web-сервера.

В качестве «тонкого» клиента в этой архитектуре выступает программа-модуль расширения Web-сервера. Сервер приложений может обеспечивать взаимодействие с Web-серверами и серверами БД, функционирующими на различных аппаратно-программных платформах (компьютерах различных типов и под управлением различных операционных систем). Такая архитектура является основой для интранет-сетей, создаваемых на основе существующих локальных сетей.

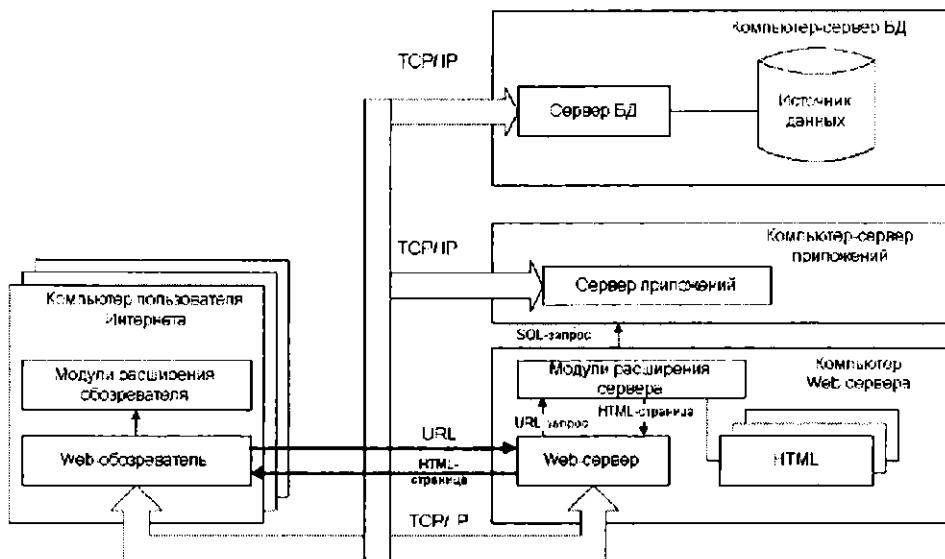


Рис. 15.7. Архитектура многоуровневого Web-приложения

Введение дополнительного уровня Web-сервера позволяет публиковать информацию из БД локальных сетей в сети Интернет, получать информацию от других интранет-сетей или Web-узлов. Кроме того, при частичной или полной реорганизации внутренней архитектуры локальных сетей появляется возможность использовать преимущества сетей интранет, касающиеся упрощения дополнительного подключения новых пользователей и администрирования локальной сети.

Отметим, что в некоторых архитектурах информационных систем Web-сервер может структурно объединяться с сервером приложений. В этом случае программные средства, входящие в состав модуля расширения, выполняют роль сервера приложений.

Основные достоинства многоуровневой архитектуры Web-приложений заключаются в следующем:

- разгрузка Web-сервера от выполнения части операций, перенесенных на сервер приложений и уменьшение размера модулей расширения сервера на основе разгрузки их от лишнего кода;
- обеспечение более гибкого межплатформенного управления между Web-сервером и сервером БД;
- упрощение администрирования и настройки параметров сети — при внесении изменений в программное обеспечение или конфигурацию сервера БД не нужно вносить изменения в программное обеспечение Web-сервера.

При функционировании Web-приложений с использованием многоуровневой архитектуры сохраняется возможность параллельной работы Web-обозревателей и клиентских приложений БД (рис. 15.8).

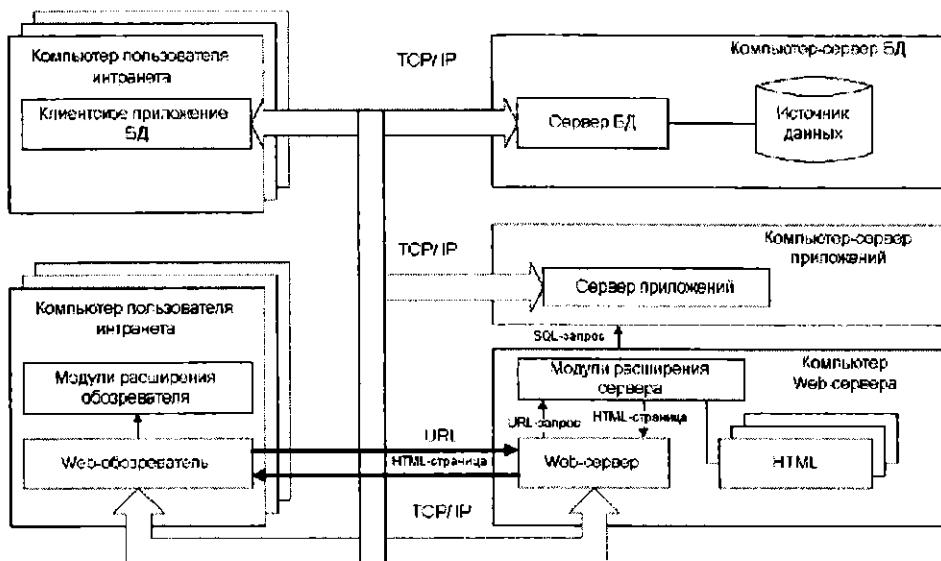


Рис. 15.8. Архитектура смешанного Web-приложения

В этом случае говорят об архитектуре смешанного Web-приложения. При такой архитектуре Web-приложения и клиентские приложения БД могут параллельно получать доступ к источнику БД.

Web-приложения на основе CORBA

Архитектура современных Интернет-приложений — это трехзвенная модель с использованием клиент/серверной архитектуры и интерфейса CGI. Однако использование CGI-интерфейса для объектно-ориентированных клиентов Java не эффективно, из-за того, что этот интерфейс достаточно медленный и не обладает требуемой гибкостью. Отметим, что попытки некоторых фирм-разработчиков программного обеспечения серверов оживить CGI, внедряя в него серверные API, являются тупиковыми.

Одним из перспективных направлений развития интранет-сетей является использование технологии CORBA (Common Object Request Broker Architecture — общая архитектура брокеров объектных запросов) в сочетании с технологией Java-апплетов. CORBA представляет собой шину (интерфейс) распределенных объектов с открытыми стандартами, используемую в клиент/серверных системах. Эта технология явилась результатом работы консорциума OMG

(Object Management Group – группы управления объектами). Единственной конкурентоспособной технологией, обладающей аналогичными возможностями, является технология DCOM (Distributed Common Object Model – распределенная модель объектов общего применения), разработанная фирмой Microsoft.

Технология Java предполагает большую гибкость при разработке распределенных приложений, но в полной мере не поддерживает технологию клиент/сервер. Интерфейс CORBA позволяет обеспечить связь переносимых приложений Java и объектов CORBA. Технология объектов CORBA предназначена для использования в Web-приложениях вместо CGI-интерфейса.

В результате объединения Java-апплетов и CORBA-интерфейса появилось новое понятие — объектная модель Web, означающая использование объектных моделей различных интерфейсов (модели CORBA, ADO и др.) при построении Web-приложений. Архитектура такого многоуровневого клиент/серверного Web-приложения, построенного на основе технологии CORBA и Java приведена на рис. 15.9.

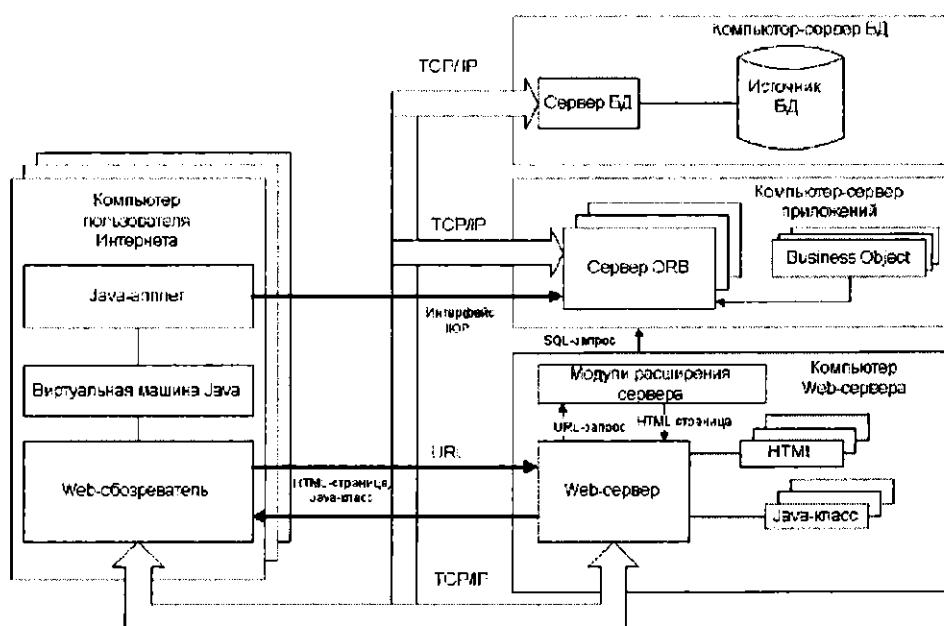


Рис. 15.9. Архитектура многоуровневого Web-приложения на основе технологии CORBA

На первом уровне находится клиентское приложение — обозреватель. В обозревателе выполняется клиентский Java-апплет, из которого может осуществляться обращение к объектам CORBA.

На втором уровне находится Web-сервер, обрабатывающий HTTP-запросы и CORBA-вызовы клиентских приложений.

На третьем уровне находится сервер-приложений. В его роли могут выступать серверы ORB (Object Request Broker – посредник запросов объектов) или распределенные объекты CORBA, функционирующие как серверы приложений промежуточного звена и выполняющие прикладные функции и набор компонентных сервисов (услуг). Серверы ORB являются унифицированными фрагментами программы, используемыми в распределенных приложениях в качестве связующего звена между клиентскими приложениями и сервером.

Объекты CORBA взаимодействуют с серверами БД последнего уровня, используя, например, SQL в случае реляционных БД. Кроме того, объекты CORBA на сервере могут взаимодействовать и друг с другом. В основе механизма взаимодействия между объектами CORBA лежит протокол IIOP (Internet Inter-ORB Protocol – Интернет-протокол взаимодействия ORB). Протокол IIOP основывается на протоколе TCP/IP с добавленными компонентами обмена сообщениями и функционирует как общий опорный протокол при организации взаимодействия серверов ORB и объектов CORBA. В дополнение к IIOP в технологии CORBA используются ESIOPIP-протоколы (Environment-Specific Inter-ORB Protocols – зависящие от среды протоколы взаимодействия ORB), которые применяются в специализированных сетевых средах.

Java-клиент может непосредственно взаимодействовать с объектом CORBA, используя Java ORB. При этом серверы CORBA замещают уровень HTTP-сервера и выступают в качестве программного обеспечения промежуточного уровня, обеспечивая взаимодействие между объектами (object-to-object). Интерфейс CORBA ПОР функционирует в сети Интернет так же, как и протокол HTTP.

Протокол HTTP в этом случае используется для загрузки Web-документов, аплетов и графики, CORBA используется для организации с помощью Java-апплетов клиент/серверных приложений.

Серверный компонент CORBA предоставляет «настраиваемый» интерфейс, который можно конфигурировать с помощью визуальных средств. CORBA-объект обладает определенными функциональными возможностями, реализует инкапсуляцию свойств и методов и генерируемых объектами событий. Можно создавать целые ансамбли объектов, «стыкуя» выходные события с входными методами. Разработка таких визуальных объектов поддерживается средствами быстрой разработки приложений – RAD (Rapid Application Development). В частности, CORBA-объекты поддерживаются в системах RAD C++Builder, JBuilder и Delphi.

На последнем четвертом уровне размещается сервер баз данных или другой источник данных, то есть практически любой источник информации, к

которому CORBA может получить доступ. Сюда входят процедурные мониторы транзакций (TP Monitors), MOM (Message-Oriented Middleware – промежуточное программное обеспечение, ориентированное на обмен сообщениями), ODBMS (ODBMS- объектные СУБД), электронная почта и т. д.

В настоящий момент интерфейсы CORBA/HTTP поддерживается почти всеми серверными платформами, включая Unix, NT, OS/2, NetWare, MacOS, OS/400.

Рассмотрим механизм функционирования Web-приложения при использовании технологии Java-апплетов и объектов CORBA.

Для начала работы с Web-приложением в Web-обозреватель загружается главная HTML-страница, которая содержит встроенные апплеты Java. При этом Java-апплеты и используемые в апплете Java-классы подгружаются с Web-сервера при открытии HTML-страницы. Причем Web-обозреватель формирует запрос Web-серверу на поиск Java-апплета или требуемого Java-класса. Web-сервер находит апплет и загружает его в обозреватель в форме байт-кода. Web-обозреватель при загрузке апплета сначала запускает систему безопасности реального времени Java. Для вызова апплетом серверных объектов CORBA используется IDL-сгенерированный клиентский стаб (Interface Definition Language – пассивный язык написания интерфейсов), который позволяет вызывать объекты сервера ORB или может использоваться интерфейс динамических вызовов CORBA DII (Dynamic Invocation Interface) для генерации запроса к серверу при выполнении апплета.

Объединение технологий CORBA, Java и Интернета обеспечивает приводимые ниже достоинства.

- Масштабируемость и устойчивость Web-приложения, заключающиеся в том, что серверы Web и ORB могут взаимодействовать с использованием CORBA ORB. При этом объекты ORB могут выполняться на нескольких серверах для обеспечения баланса нагрузки (load-balancing) серверов для входящих клиентских запросов. ORB может отправить запрос первому доступному объекту, а также увеличить число доступных объектов по необходимости. CORBA позволяет объектам сервера действовать последовательно, используя транзакции и связанные сервисы CORBA. В отличие от технологии CORBA, интерфейс CGI при большом количестве запросов не имеет возможности распределить нагрузку между несколькими процессами или процессорами.
- Гибкость архитектуры CORBA позволяет клиентам непосредственно вызвать методы на сервере. Клиенты передают параметры напрямую, используя прекомпилированные стабы, или генерируют их во время выполнения апплета, используя сервисы динамических вызовов CORBA DII. Можно вызывать на сервере любой метод, определенный с помощью IDL, а не только один метод, описанный посредством HTML, можно передавать любые типизированные параметры вместо обычных строк.

- CORBA расширяет возможности Java по взаимодействию с распределенными объектами. Так как Java-апплеты не могут взаимодействовать сквозь все адресное пространство, используя вызовы удаленных методов, то для Java-апплетов не существует простого способа вызвать метод на удаленном объекте. CORBA позволяет Java-апплетам взаимодействовать с другими объектами, написанными на различных языках, преодолевая адресное пространство и сети. CORBA обеспечивает богатый набор сервисов распределенных объектов (метаданные, транзакции, безопасность, именование, коллекции и т.д.), которые расширяют Java.
- CORBA расширяет объектную модель Java для распределенной среды и позволяет Java-апплетам вызывать широкий спектр определенных на IDL операций на сервере. В противоположность этому, клиенты HTTP ограничены небольшим набором операций. Приложения серверной части — это обычные объекты CORBA. Следовательно, они доступны в любой момент времени. Нет необходимости проходить через издержки обращения к CGI-сценариям для каждого вызова.
- CORBA предоставляет средства для создания трехзвенной архитектуры клиент/сервер. Кроме того, технология CORBA разгружает код Java-апплетов, определенные компоненты могут быть распределены в среде клиент/сервер. Клиентская часть апплета может оставаться маленькой, что сокращает время загрузки апплета.

Кроме технологии CORBA, для расширения возможностей применения Web могут использоваться следующие конкурирующие технологии: сокеты, DCOM с ActiveX и RMI (Remote Method Invocation — механизм вызова удаленных методов) другие технологии. Тем не менее, технология CORBA остается лидером среди всех интерфейсов распределенных объектов благодаря хорошим архитектурным и функциональным возможностям, устойчивости в работе и легкости в настройке.

В частности, технология CORBA по сравнению с ближайшим ее конкурентом — технологией DCOM обеспечивает следующие преимущества:

- полная и корректная реализация поддержки различных ОС;
- CORBA реализована целиком на Java, в связи с этим она хорошо интегрируется с Java;
- легкость конфигурирования и настройки серверов и клиентов CORBA;
- корректная реализация вызовов распределенных методов;
- полнота функциональной реализации динамического наследования и поддержки метаданных;
- более высокая (более чем на 20%) производительность приложений;
- корректная реализация механизма транзакций;
- корректная реализация долговременных, или сохраняемых объектных ссылок;
- поддержка CORBA-интерфейсом URL-имен;
- открытый стандарт CORBA-интерфейса.

Таким образом, CORBA обеспечивает инфраструктуру распределенных объектов, что позволяет приложениям распространяться через сети, языки, границы компонентов и операционные системы. Java обеспечивает инфраструктуру переносимых объектов, которые работают на всех основных операционных системах. CORBA дает независимость от сетей, а Java — независимость от реализации.

Для организации связи программных расширений Web-сервера с БД используются современные интерфейсы доступа к данным OLE DB, ADO и ODBC. Эти интерфейсы являются промежуточным уровнем между источником данных и приложением, в качестве которого выступают программные расширения Web-сервера. Рассмотрим особенности архитектуры Web-приложений, использующих интерфейсы доступа к данным OLE DB, ADO и ODBC.

Характеристика интерфейсов OLE DB, ADO и ODBC

Современные интерфейсы доступа к источникам данным OLE DB, ADO и ODBC, внедряемые фирмой Microsoft, позволяют осуществлять доступ к источникам различных данных однообразным способом. В основе новой технологии доступа к данным лежит интерфейс OLE DB, позволяющий связывать и встраивать объекты из любых источников данных. Напомним, что интерфейс OLE DB является универсальной технологией для доступа к любому источникам данных через стандартный интерфейс COM. Интерфейс OLE DB основан на механизме сервис-провайдеров (поставщики данных), находящихся на высоком уровне абстракции над физическим форматом данных.

OLE DB-провайдер реализует интерфейс доступа OLE DB поверх конкретного сервис-провайдера данных. Интерфейс доступа OLE DB позволяет вводить масштабируемость, заключающуюся в возможности поддерживать многоуровневую систему OLE DB-провайдеров, когда OLE DB-провайдер может находиться поверх группы OLE DB-провайдеров или сервис-провайдеров. Интерфейс OLE DB позволяет стандартизовать программный код, использующийся для доступа к различным источникам данных.

Архитектура Web-приложений, использующих интерфейсы OLE DB, ADO и ODBC, приведена на рис. 15.10.

Интерфейс ADO находится на более высоком уровне абстракции, чем интерфейс OLE DB. Он реализован в виде иерархической модели объектов для доступа к различным OLE DB-провайдерам данных. В модель ADO входит набор объектов, которые обеспечивают соединение с провайдером данных, создание SQL-запроса к данным, создание набора записей на основе запроса и др.

Особенность функционирования Web-приложений, использующих интерфейс ADO, заключаются в том, что обозреватель может извлекать информацию из любого источника данных, находящегося в сети Интернет, заранее не имея представления о логической структуре, типе и физическом формате источника данных. То есть появляется возможность публиковать требуемую информацию в Интернете, не показывая внутреннюю структуру данных.

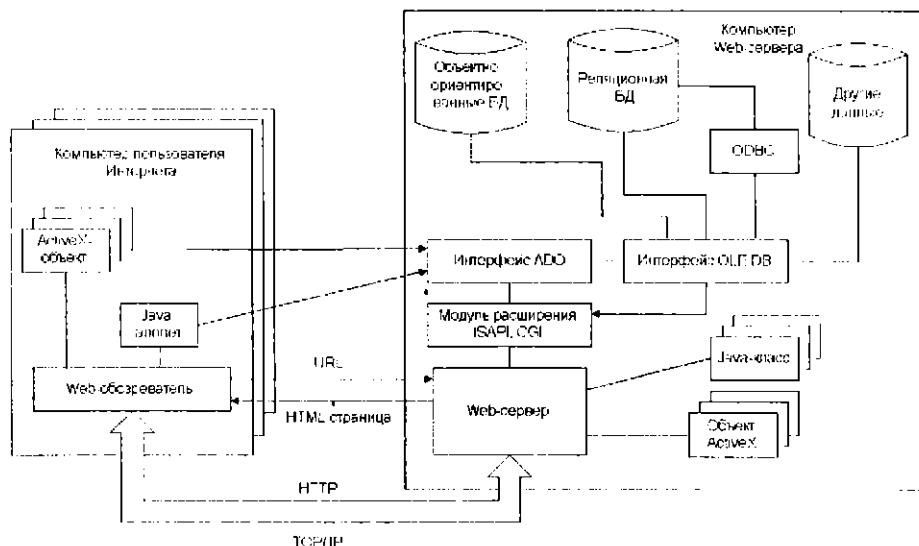


Рис. 15.10. Архитектура Web-приложений, использующих интерфейсы OLE DB, ADO и ODBC

Таким образом, интерфейс ADO позволяет стандартизовать все существующие интерфейсы для доступа к любым данным в интранет/Интернет сетях. Он объединяет все существующие интерфейсы и предоставляет однообразный способ для доступа к любому источнику данных через любой интерфейс.

15.3. Обзор Web-серверов

В настоящий момент развитие технологий Интернета идет очень быстрыми темпами. Появляется очень много производителей программного обеспечения для Интернета, в том числе производителей Web-серверов. Дадим характеристику наиболее распространенным Web-серверам, используемым в корпоративных сетях и в «домашних» компьютерах, которые могут использоваться для публикации информации из БД. Определим понятие Web-сервера. Web-сервер – это программное средство, установленное на Web-узле глобальной или корпоративной сети и позволяющее пользователям сети получать доступ к гипертекстовым документам, расположенным на этом Web-узле. Иногда под Web-сервером понимают программное обеспечение Web-сервера и аппаратное обеспечение – компьютер, на котором Web-сервер установлен.

В общем случае программное обеспечение Web-сервера может устанавливаться на компьютеры общего назначения, предназначенные для решения различ-

цих задач, не обязательно связанных с технологиями Интернета. Поэтому более корректно использовать понятие Web-сервера для обозначения только программного обеспечения Web-сервера, а компьютер с операционной системой и сетевой структурой называть средой работы Web-сервера, или платформой.

Отметим, что Web-серверы используются для следующих целей:

- создание корпоративных сетей интранет на основе принципов Интернет-сетей, многоуровневой архитектуры и клиент/серверных технологий;
- подключение корпоративных сетей интранет к Интернету для получения доступа к предоставляемым в нем услугам;
- публикация информации из корпоративных сетей интранет, в том числе и содержимого БД из информационных систем, функционирующих в среде интранет;
- распространение собственной информации, находящейся на домашнем компьютере, создание собственного сайта с помощью домашнего компьютера.

В настоящее время в Интернете функционирует большое число типов серверов, используемых для обеспечения различных функций. Кроме того, существует большое число однотипных серверов, разработанных различными производителями.

В Интернете существует ряд источников информации о серверах, содержащих ежемесячные обзоры, всего, что касается Web-серверов. Например, один из наиболее популярных источников информации о статистики использования Web-серверов находится в Интернете на узле Netcraft по адресу "<http://www.netcraft.com/survey>", который содержит самую свежую информацию о серверах и платформах, используемых в Интернете, и список узлов, содержащих информацию об основных Web-серверах.

Еще один из лучших источников сводной информации находится на узле iWORLD компании Meckermedia (<http://www.iworld.com>), на котором можно найти журналы WebCompare, ServerWatch и Web Servers Feature Chart.

В таб. 15.4 по данным, опубликованным на узле netcraft, представлены сведения по трем наиболее популярным серверам. Как следует из приведенной таблицы, наибольшее распространение имеют Web-серверы Apache и Microsoft. Причем имеет место тенденция небольшого снижения со временем процента этих Web-серверов от общего числа сайтов в Интернете.

Таблица 15.4.
Популярные Web-серверы

Название Web-сервера	Число сайтов в Интернете осенью 2000	Процент от числа сайтов в Интернете	Изменение процента от числа сайтов за квартал
Apache	12705194	60,02	-1,64
Microsoft	4140977	19,56	-0,07
Netscape-Enterprise	1472689	6,96	-0,05

На выбор сервера большое влияние оказывает «платформа», на которой работает Web-сервер. Анализ различных источников показывает, что в качестве узлов Web могут работать компьютеры любых типов с необходимыми техническими характеристиками, касающиеся ресурсов и в целом производительности компьютера. Аппаратное обеспечение, используемое в платформах серверов, может включать большинство типов компьютеров, используемых на сегодня.

Активно используемых типов операционных систем намного меньше, чем типов компьютеров. Анализ статистики показывает, что для высокопроизводительных объемных узлов наиболее часто используется операционная система Unix (около 80% Web-серверов работают под ее управлением), для средне- и низкопроизводительных узлов чаще всего используется Windows NT (менее 20% Web-серверов работает под ее управлением).

Операционные системы Web-серверов

В сети Интернет в основном используется несколько операционных систем. Наиболее популярными среди Web-серверов являются Unix-подобные операционные системы. Дадим краткую характеристику основным операционным системам, используемым в качестве платформ Web-серверов.

Операционная система Unix (Unix-подобные операционные системы) получила наибольшее распространение в среде Интернет по следующим причинам:

- Unix применяется для значительно большего количества платформ, чем другие операционные системы. Она распространяется в исходных кодах, поэтому легко может быть перекомпилирована для любой аппаратной платформы;
- Unix раньше других начала применяться в Интернете;
- Unix включает большое количество услуг, ориентирована на работу с наибольшим количеством процессоров, адресов IP;
- Unix является более устойчивой при функционировании в загруженных сетях.

Однако эта система является самой сложной для изучения и конфигурирования из всех операционных систем.

Операционные системы Windows 2000 Server и Windows NT Server широко распространены, используются в качестве платформы Web-серверов и являются единственными реальными конкурентами для системы Unix. Отметим, что Windows 2000 Server представляет собой доработанный вариант Windows NT Server и включающий достоинства Windows 98. Основное преимущество Windows 2000/NT Server заключается в легкости настройки и освоении работы в среде этих операционных систем. Для начинающих пользователей лучше начать работу в Web с одной из этих операционных систем.

Для Web-узла с малой или средней нагрузкой операционная система Windows 2000/NT Server фирмы Microsoft работает достаточно надежно. Об этом свидетельствует тенденция роста в процентном соотношении числа Web-серверов, использующих Windows 2000/NT Server. Отметим, что другие операционные системы Windows 9X фирмы Microsoft не зарекомендовали себя как надежные платформы для Web-узла.

В Windows 2000 расширены средства поддержки операционных систем. Так, Windows 2000 позволяет организовывать взаимодействие с Windows NT Server 3.51 и 4.0, поддерживает клиентов с операционными системами Windows 3.x, Windows 95, Windows 98 и Windows NT Workstation 4.0, с большими и средними ЭВМ с помощью шлюзов транзакций и очередей. Файловый сервер для Macintosh позволяет клиентам Macintosh организовывать общий доступ к файлам и использовать общие ресурсы Windows 2000/NT Server.

Тем не менее операционная система Windows 2000/NT Server не обеспечивает требуемую гибкость при администрировании и расширении возможностей Web-узла.

Отметим, что тенденция увеличения использования Windows 2000/NT будет сохраняться за счет входления в сеть Интернет большого количества пользователей Intel-компьютеров. Windows-ориентированные серверы могут устанавливаться и настраиваться автоматизированно, в отличие от Unix-ориентированных Web-серверов, которые с трудом поддаются настройке.

Операционная система Mac OS используется пользователями Macintosh, работающими в Web. В настоящее время эта операционная система значительно уступает по популярности Windows NT и Unix ввиду того, что система имеет ограниченные возможности по взаимодействию с динамическими узлами и не приспособлена к режиму повышенной нагрузки. Mac OS не является лучшим выбором операционной системы для Web-сервера, но остается популярной среди пользователей Macintosh.

Приведем основные отличия операционных систем Unix и Windows 2000/NT Server.

- В Unix более гибко реализована масштабируемость, в этой операционной системе могут использоваться кластерные технологии.
- В Unix реализована избыточная отказоустойчивость. На основе Unix построены мэйнфреймы — самые устойчивые аппаратно-программные комплексы.
- Общая безопасность системы с точки зрения защиты информации операционной системы Unix превосходит Windows 2000/NT Server.
- Unix имеетстроенную поддержку многопользовательского интерфейса, которая отсутствует у Windows NT. К компьютеру, на котором установлена операционная система Unix, могут быть подключены алфавитно-цифровые терминалы, с которых можно работать с графической многопользовательской оболочкой операционной системы Unix.

- Операционная система Unix имеется на всех аппаратных платформах. Сама операционная система Unix и входящие в ее состав приложения распространяются в исходных кодах. Для переноса Unix или любого Unix-приложения на другую аппаратную платформу достаточно перекомпилировать на требуемой платформе операционную систему Unix или Unix-приложение.
- Web-сервер для операционной системы Unix остается более устойчивым к сбоям, объему нагрузки и обеспечивает более высокую безопасность Web-узла.
- Сервер баз данных для операционной системы Unix, благодаря кластерной технологии и лучшей масштабируемости, чаще используется в качестве мощного сервера баз данных.
- Для сервера приложений лучше подходит Windows 2000/NT Server, так как эта операционная система обеспечивает более полное использование возможностей Windows-подобных приложений.
- Windows 2000/NT Server обладает удобным графическим интерфейсом, что значительно упрощает настройку, администрирование и процесс обучения персонала работе на Web-узле.
- Администрирование реализовано в Windows 2000/NT Server с использованием автоматизированных средств и Мастеров.
- В операционной системе Unix реализован более широкий набор утилит, необходимых для удаленного администрирования.

Дадим обзор наиболее популярных Web-серверов.

Сервер Apache

Под управлением Apache работают более 6 миллионов серверов Интернет (на февраль 2000 г.). Они полностью отлажены и протестированы разработчиками и пользователями. Группа разработчиков Apache придерживается строгих стандартов в отношении выпуска новых версий их сервера. Когда обнаруживаются ошибки в работе сервера, компания Apache Development Group выпускает корректирующие файлы или новые версии продукта. Эта компания является международной организацией добровольцев, разработавших данный программный продукт для некоммерческого распространения среди широкого круга пользователей. Созданный под покровительством компании Apache Digital Corporation, проект Web-сервера Apache развивался как ветвь NCSA httpd проекта одного из самых первых наиболее эффективных из уже давно существующих серверов сети Интернет.

Само название «Apache» созвучно слову «A PAtCHy server» (сервер с доработками — «patch files»). Одновременно это — просто красивое название, связанное с американским индейским племенем Apache, известным своим военным мастерством и неутомимостью.

В сравнении с другими серверами, Apache показал себя более устойчивым, более быстрым, имеющим более широкий набор функций и возможностей. Кроме того, Web-сервер Apache характеризуется открытой архитектурой, за-

ключающейся в том, что этот сервер распространяется в исходных кодах и используется гибкая архитектура построения сервера, позволяющая легко наращивать дополнительные возможности.

Его безусловное доминирование на рынке объясняется тем, что сервер был разработан для самой популярной платформы UNIX. Хотя сервер Apache распространяется бесплатно, но организация, разработавшая и обслуживающая этот мощный пакет (см. сервер www.apache.org), обеспечивает свое функционирование с помощью пользователей, которые спонсируют его развитие и сопровождение.

Используя открытый код Apache, разработчик может создавать собственные конфигурации сервера, компилируя внесенные в код изменения. Apache имеет модульную структуру, то есть в его состав входит набор модулей, которые служат для обеспечения требуемых функций сервера и могут быть динамически включены в конфигурацию даже во время активной работы сервера. Сервер Apache позволяет использовать CGI-сценарии, написанные на Perl или PHP.

Приведем основные особенности функционирования сервера Apache.

- Является мощным, гибким, HTTP/1.1-совместимым сервером.
- Поддерживает современные протоколы.
- Имеет легко перестраиваемую конфигурацию с возможностью установления дополнительных функций (модулей) от сторонних производителей.
- Может быть сконфигурирован с использованием модулей API.
- Снабжается полным исходным текстом и поступает с бесплатной лицензией без ограничений.
- Работает под управлением популярных операционных систем Windows NT/9x, Netware 5.x, OS/2 и большинства версий Unix.
- Поддерживает введение отчетной документации об ошибках и файлы коррекции.

Сервер Apache поддерживает следующие функции:

- доступ к базам данных, используемым для аутентификации, то есть возможность установки защищенных паролем страниц с огромным количеством уполномоченных пользователей без перегрузки сервера;
- настройку реакции сервера на ошибки и сбои, заключающуюся в возможности устанавливать файлы или даже сценарии CGI, используемые сервером при возникновении ошибки (например, установка сценариев, позволяющих обрабатывать около 500 ошибок сервера, вести непрерывную диагностику и устранять неполадки по желанию пользователя);
- автоматическая обработка HTML-данных с изменяющейся структурой и модификация их для удобного представления информации клиенту;
- поддержка виртуальных хостов, заключающаяся в возможности настройки нескольких «домашних хостов», что позволяет серверу различать запросы, сделанные по различным IP-адресам. Apache также предоставляет возможность динамически настраивать функции «главного» виртуального хоста;

- генерация информации о настройках в удобном для пользователя формате;
- формирование на большинстве архитектур Unix Apache так называемых журналов учета (log файлов), причем их количество кратно числу виртуальных хостов, и работа DNS корректируется без приостановки функционирования.

Отметим, что для сервера Apache отсутствуют официальное техническое обслуживание и поддержка. Тем не менее, для этой программы можно найти большое количество информации или советов. Например, список известных сбоев можно найти на Web-узле, а со службой поддержки от независимых разработчиков можно связаться через список рассылки Apache comp.infosystems.www.servers.unix, через коммерческие службы, например, Cygnus (<http://www.cygnus.com/product/idk/apache>) или ежемесячный дайджест от разработчиков Apache. Еще одним неплохим источником технической информации является Apache Week.

В перспективе для сервера Apache планируется разработать графический пользовательский интерфейс и окончательно адаптировать версию этого сервера для работы на платформе Windows NT/9X. Кроме того, основные направления политики внедрения сервера Apache в Интернете остаются прежними, а именно открытость архитектуры, тесная обратная связь с пользователями и поддержка последних версий протокола HTTP.

Microsoft Internet Information Server

Ближайшим конкурентом Apache является Microsoft Internet Information Server (MIIS), входящий в состав системы Windows 2000/NT Server. Согласно опросу, проводимому компанией Netcraft (табл. 15.4), под управлением MIIS работает примерно в три раза меньше серверов от количества, обслуживающего Apache, то есть этому серверу отдают предпочтение около 20 процентов пользователей (Apache используют почти более 60 процентов пользователей). Он обладает многими новыми функциональными возможностями, среди которых создание нескольких Web-узлов, новые средства администрирования, работа в режиме сервера новостей.

Еще одной особенностью стали определяемые пользователем специальные группы Интернета, где можно размещать различные ресурсы, например принтеры, с целью упрощенного обращения к ним и просмотра.

Службы MIIS позволяют использовать масштабируемые Web-приложения, а также публиковать информацию из информационных систем в Интернете. Службы MIIS включают поддержку страниц ASP, которые являются средой создания серверных сценариев для разработки динамических интерактивных приложений Web-серверов. Они позволяют разработчикам объединять нужным образом страницы в формате HTML, команды сценариев и компоненты COM для создания мощных и гибких Web-приложений.

Упрощенной версией MIIS является Microsoft Personal Web Server (PWS), который предназначен для работы в качестве настольного Web-сервера для распространения информации с домашнего компьютера в сеть.

К недостаткам MIIS относят то, что этот сервер разработан в основном для платформ Windows 2000/NT и 9Х. Используется сервер, как правило, с операционной системой Windows 2000/NT Server, так как система Windows 2000/NT имеет намного более высокий уровень устойчивости к сбоям и общей стабильности при работе по сравнению с операционными системами Windows 9Х. Анализ статистических данных и отзывов пользователей показывает, что общая производительность и надежность MIIS на платформе Windows 2000/NT Server близка к среднему показателю для сервера Apache.

MIIS 4.0 является встроенным в Windows NT 5.0 сервером. MIIS поставляется бесплатно как часть пакета NT Server.

Microsoft Internet Information Server позволяет так же расширять возможности сервера, как и сервер Apache. Используя интерфейс COM, можно подключать различные языки создания сценариев для этого Web-сервера. Но в этом случае интерпретаторы сценариев не входят непосредственно в состав сервера, что несколько замедляет обработку сценариев.

Наибольшее отличие между Apache и MIIS заключается в возможности изменять конфигурацию сервера Apache без остановки его работы, изменения в текстовом режиме файлы настройки, что позволяет динамически включать и выключать необходимые модули. Однако графический интерфейс MIIS развит гораздо лучше, в то время как аналогичный модуль Apache все еще в стадии разработки. Это объясняется тем, что первоначально Apache разрабатывался для Unix-подобных систем с интерфейсом в виде командной строки.

Среди недостатков MIIS можно выделить недостаточный уровень безопасности дисков с файловой системой NTFS, в результате которой возникает зависимость от сети и системы безопасности Windows NT.

Положительным фактом для MIIS является возможность получения технической поддержки этого сервера. В интерактивном режиме предоставляется вся информация, касающаяся установки и обслуживания. Дополнительно в пакете MIIS имеются отдельные файлы помощи по утилитам.

Серверы Netscape Enterprise

На третьем месте в рейтинге использования серверов находится сервер компании Netscape, которая осуществляет продажу разнообразных серверных продуктов (Communications, Commerce и Enterprise). Сервер Netscape Enterprise используют около 7% Web-узлов. Этот сервер уверенно конкурирует с основными типами серверов.

Достоинством этого сервера является то, что фирма Netscape разработала версии сервера, которые работают в системах Windows 95, NT и UNIX.

Сервер Netscape характеризуется простотой в использовании и надежностью в работе, «интегрированностью» высокопроизводительных функций в интерфейсе пользователя.

Информацию по программным продуктам Netscape можно найти по адресу www.netscape.com.

15.4. Использование Personal Web-server

Microsoft Personal Web Server (PWS) 4.0 является настольным Web-сервером. При условии подключения к сети интранет распространение документов между клиентами можно выполнять непосредственно со своего компьютера. PWS позволяет разрабатывать и публиковать свою личную основную страницу. Упрощается также проверка своего Web-узла перед его передачей поставщику услуг Интернета. Personal Web Server является упрощенным вариантом сервера Microsoft Internet Information Server. Personal Web Server предназначен для передачи информации в формате HTML-страниц с использованием протокола HTTP.

Этот сервер реализует следующие основные возможности, которые предоставляет сервер Microsoft Internet Information Server:

- создание собственного сайта на своем компьютере;
- публикацию Web-страниц в Интернете, в том числе публикацию БД в сетях интранет;
- возможность использования технологии объектов ActiveX;
- передачу файлов на основе службы FTP;
- реализацию технологии IDC/HTX-страниц на основе компонента Internet DataBase Connector, функционирование которого обеспечивается библиотекой httpodbc.dll.

Для начала работы с сервером Personal Web Server необходимо выполнить следующие действия:

- установить Personal Web Server на компьютере;
- запустить установленный Personal Web Server на компьютере;
- проверить работоспособность Personal Web Server на подключенном к сети Интернет компьютере;
- настроить параметры Personal Web Server (выполнить конфигурирование и администрирование сервера);
- подготовить в каталоге, содержащем начальную страницу Вашего сайта (по умолчанию для Personal Web Server — это каталог C:\Webshare\wwwroot), все необходимые HTML-страницы, входящие в состав Web-приложения, обеспечивающего работу сайта.

Рассмотрим выполнение указанных действий.

Установка сервера

Установка Personal Web Server может осуществляться двумя способами в зависимости от того, входит Personal Web Server в дистрибутив Windows 98 или используется отдельный дистрибутив этого сервера. В первом случае Personal Web Server может быть установлен при общей установке системы или в варианте дополнительной установке компонентов операционной системы. При общей установке системы необходимо выбрать режим установки «Для опытных пользователей» (детальный выбор) программных компонентов операционной системы Windows 98. Для этого в группе компонентов «Средства Интернет» нужно определить состав компонентов, нажав на кнопку «Состав». При появлении окна выбора компонентов группы «Средств Интернет» нужно отметить Personal Web Server, находящийся в верхней позиции.

При установке в варианте дополнительных компонентов операционной системы без установки самой операционной системы необходимо через кнопку «Пуск» выбрать пункт меню – «Настройка», затем запустить «Панель управления» и в «Панели управления» выбрать «Установка/удаление программ», затем выбрать вкладку «установка Windows». Далее отметить Personal Web Server в окне выбора компонент группы «Средства Интернет» (рис. 15.11). Отметим, что не все дистрибутивы Windows 9X содержат дистрибутив Personal Web Server.

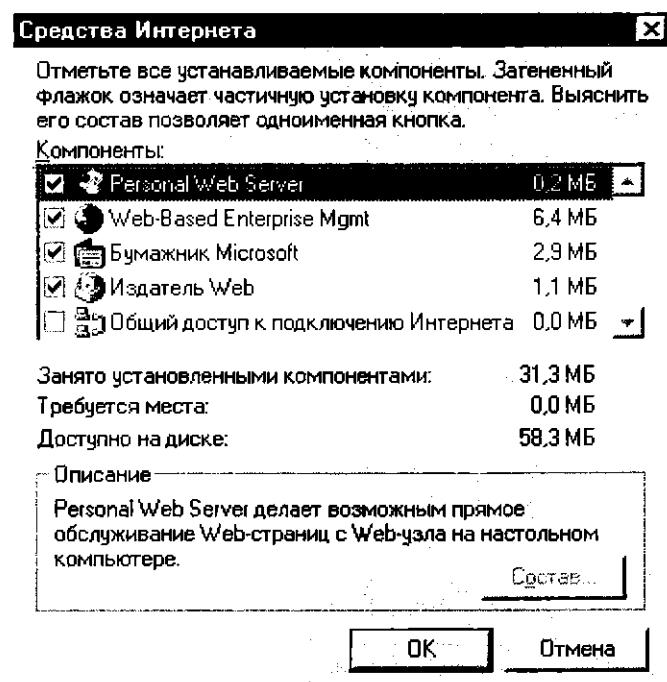


Рис. 15.11. Группа «Средства Интернета» в окне установки компонентов Windows

При установке Personal Web Server с отдельного дистрибутива от пользователя требуется только запустить программу установки для этого сервера.

Для установки Microsoft Personal Web Server с установочного диска Windows 98 достаточно выполнить следующую последовательность действий:

- вставить компакт-диск Windows 98 в дисковод;
- нажать кнопку **Пуск** и выбрать команду **Выполнить**;
- в поле Открыть ввести: `x:\add-ons\pws\setup.exe` (`x` – имя дисковода для компакт-дисков);
- нажать кнопку **OK**;
- выполнить указания программы установки Personal Web Server.

Администрирование сервера

Для администрирования Personal Web Server нужно открыть «Панель управления» и запустить приложение-менеджер Personal Web Server, управляющее свойствами сервера. Кроме того, существует возможность запуска приложения-менеджера с помощью иконы Personal Web Server, расположенной в крайней правой части панели задач (по умолчанию присутствует в нижней части рабочего стола и содержит кнопку «Пуск»). После запуска приложения-менеджера появляется окно свойств Personal Web Server (рис. 15.12), в котором имеются следующие вкладки: **General**, **Startup**, **Administration**, **Services**.

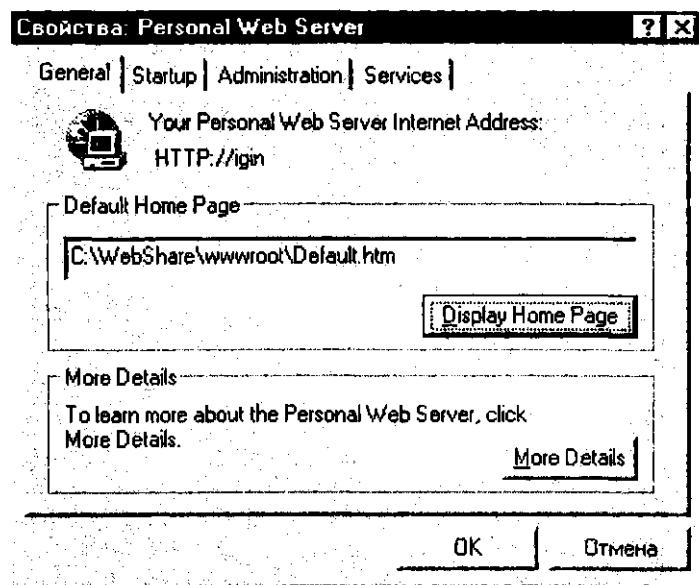


Рис. 15.12. Окно свойств Personal Web-Server

На вкладке **General** в верхней части находится информация, характеризующая адрес Personal Web Server в сети Интернет, ниже находится информационная строка, содержащая путь к домашней странице. В нашем случае путь к домашней странице, как и по умолчанию, есть C:\WebShare\wwwroot\Default.htm.

Под домашней страницей понимается начальная (первая отображаемая) страница Web-узла или раздела Web-узла, используемая для получения доступа к набору страниц, хранящемуся в домашнем каталоге или других виртуальных каталогах. Домашним каталогом является корневой каталог Web-узла, в котором находятся файлы Web-узла, доступные для пользователей сети. Виртуальный каталог – это используемое в URL-адресе имя каталога (алиас виртуального каталога), соответствующее физическому каталогу на сервере.

На вкладке **Startup** находятся кнопки Start и Stop, служащие для запуска и остановки работы сервера.

Вкладка **Services** содержит список запущенных служб и ряд кнопок. Кнопка Properties позволяет устанавливать свойства, связанные с каждой службой. Для Personal Web Server имеются службы HTTP и FTP. Для изменения параметров служб следует в списке служб выбрать требуемую службу и нажать на кнопку Properties. При этом появляется окно, содержащее параметры службы. В нем находятся кнопки, позволяющие изменять корневой каталог Web-узла и домашнюю страницу, открываемую по умолчанию. При выборе любой из кнопок открывается обозреватель, активной вкладкой в котором является вкладка Directories.

Вкладка **Administration** имеет одну кнопку, которая позволяет администрировать Personal Web Server. С помощью этой кнопки открывается обозреватель, в который загружается HTML-документ, имеющий внешний вид, представленный на рис. 15.13.

Названный HTML-документ (рис. 15.13) отвечает за администрирование Web-сервера и содержит интерактивные элементы для задания параметров и каталогов. В нем находится панель интерактивных элементов, имеющая три вкладки Service, Directories и Logging.

Основной является вкладка **Directories**, в которой находятся таблица, содержащая корневой каталог сервера, и другие виртуальные каталоги. В левой части таблицы находится локальный путь к виртуальным каталогам, правее в таблице находятся алиасы, присвоенные этим виртуальным каталогам. Виртуальный каталог представляет собой локальный каталог Web-сервера, в котором всем пользователям сети разрешен доступ к HTML-документам и исполнение файлов скриптов. В нашем случае под алиасом виртуального каталога понимают специальное имя, используемое в адресе URL после имени хоста. Алиас указывает, к HTML-документам какого виртуального каталога обращается обозреватель.

В таблице (рис. 15.13) в крайней правой колонке (Action) приведен список возможных действий: редактирование виртуального каталога, удаление и добавление нового виртуального каталога. При выборе действия Edit появляется окно, в котором находятся группы полей, позволяющие устанавливать относительный

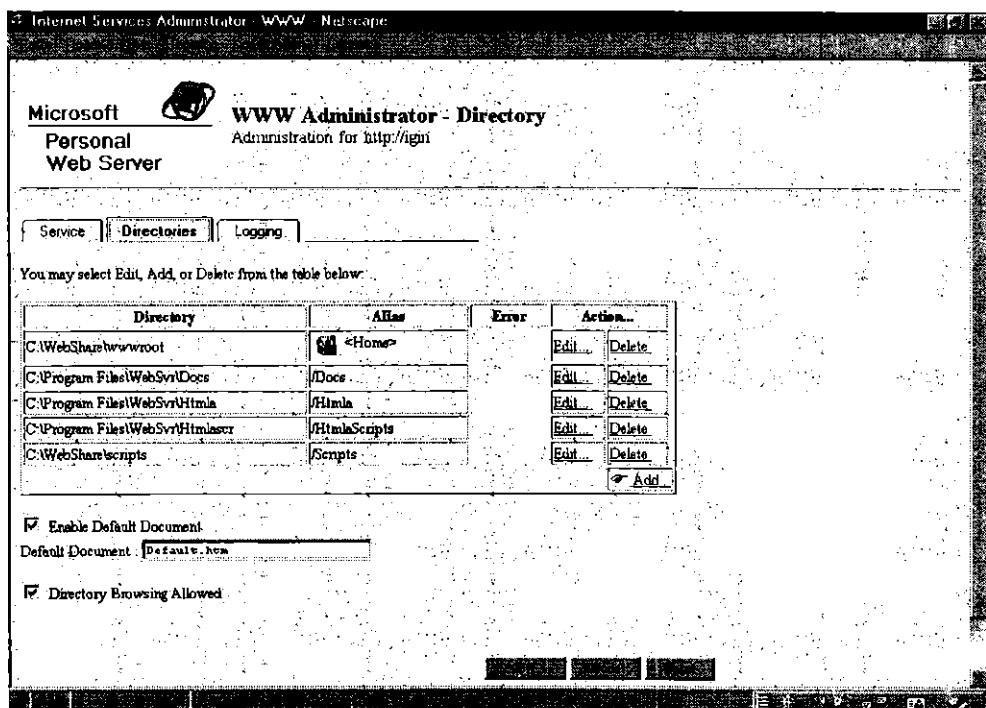


Рис. 15.13. HTML-документ администрирования Personal Web Server

путь к каталогу, имя алиаса, тип доступа к каталогу (для текстовых файлов и/или файлов сценариев) и устанавливать виртуальный каталог как домашний каталог.

15.5. Использование Microsoft Internet Information Server

MIIS (Microsoft Internet Information Server 5.0) является составной частью Windows NT Server, начиная с четвертой версии этой операционной системы, а в рабочую станцию входила его упрощенная версия — Personal Web Services.

MIIS представляет Web-службу Windows 2000 Server, использующемся для публикации информации в интранет или Интернет.

MIIS является встроенной в операционную систему Windows 2000 Server службой. Он обладает многими новыми функциональными возможностями, среди которых создание нескольких Web-узлов, новые средства администрирования, работа в режиме сервера новостей, определяемые пользователем специальные группы Интернета, где можно размещать различные ресурсы, например принтеры, с целью упрощенного обращения к ним и просмотра.

Windows 2000 Server является мощным сервером приложений баз данных, соединяющим в себе сетевую операционную систему и службы Интернета. Windows 2000 Server обеспечивает масштабируемость, поддержку больших объемов физической памяти, достаточный уровень надежности и безопасности, удобный графический интерфейс, мощный набор средств и Мастеров, автоматизирующих настройку и администрирование Web-узла.

Windows 2000 Server подходит для быстрой разработки собственного Web-узла, интенсивно работающего с базами данных, особенно в качестве сервера приложений. При проектировании структуры Web-узла с использованием MIIS на основе платформы Windows 2000 Server наиболее эффективно использовать многоуровневую схему. При этом в качестве сервера баз данных предпочтительно использовать сервер БД на основе операционной системы Unix, а Web-узел с Windows 2000/NT Server – использовать в качестве сервера приложений. Именно такая схема используется на одном из самых посещаемых Web-узле фирмы Microsoft – "<http://www.microsoft.com>".

MIIS 5.0 имеет много новых возможностей, помогающих Web-администраторам создавать масштабируемые гибкие Web-приложения.

- Средства обеспечения безопасности:
 - дополнительная проверка подлинности позволяет осуществлять безопасную проверку подлинности пользователей через прокси-серверы и брандмауэры в дополнение к обычной проверке подлинности;
 - безопасные подключения, обеспечивающие безопасный обмен информацией между клиентами и серверами;
 - шифрование передаваемой информации;
 - Мастера безопасности, упрощающие решение задач администрирования сервера.
- Средства автоматизации администрирования:
 - Мастер сертификатов Web-сервера упрощает решение задач администрирования сертификатов, например создание запросов на сертификаты и управление жизненным циклом сертификатов;
 - Мастер разрешений упрощает конфигурирование доступа к Web-узлу присвоением правил доступа виртуальным каталогам или файлам.
- Поддержка технологии Active Server Pages, которая составляет альтернативу технологиям CGI и ISAPI для доступа к базам данных и имеет ряд новых и улучшенных возможностей для повышения быстродействия и гибкости сценариев на стороне сервера.
- Более мощная защита и увеличение надежности Web-приложений. По умолчанию MIIS выполняет все приложения в общем или групповом процессе, который отделен от процессов ядра MIIS. Кроме того, остается возможность изолирования критически важных приложений, которые следует запускать вне процессов ядра MIIS и вне группового процесса.

Установка сервера

Службы MIIS 5.0 по умолчанию устанавливаются в системе Windows 2000 Server. Для переустановки MIIS, добавления или удаления компонентов MIIS нужно запустить приложение Установка и удаление программ, входящее в состав панели управления. Для этого следует нажать кнопку «Пуск», выбрать «Настройка», «Панель управления», а затем выполнить двойной щелчок на значке «Установка и удаление программ».

Далее нужно выбрать вкладку «Добавление и удаление компонентов Windows», нажать кнопку «Компоненты» и следовать отображаемым указаниям по установке, удалению и добавлению компонентов MIIS.

При обновлении системы до Windows 2000, MIIS 5.0 будет установлен по умолчанию только в случае, если в предыдущей версии Windows были установлены службы MIIS.

В процессе установки MIIS на компьютере создаются следующие каталоги для хранения данных, которые предполагается опубликовать в Интернете:

- \Inetpub – домашний каталог по умолчанию;
- \корневой каталог системы\Help\iisHelp – виртуальный каталог.

Для обеспечения безопасности рекомендуется, чтобы все диски, используемые MIIS, были отформатированы в формате NTFS. Перед установкой MIIS на компьютере должны быть установлены модули, обеспечивающие протокол TCP/IP, и служебные программы связи.

Для обеспечения возможности публикации данных поставщик услуг Интернета должен предоставить IP-адрес сервера, маску подсети и IP-адрес шлюза, используемого по умолчанию. Используемый по умолчанию шлюз является компьютером поставщика услуг Интернета, через который ваш компьютер маршрутизирует весь поток данных Интернета.

Желательно также установить на компьютер дополнительно службу DNS (Domain Name System) при планировании развертывания сети. Это необязательное условие, но оно дает пользователям возможность применять «понятные» текстовые имена вместо IP-адресов. В Интернете Web-узлы обычно используют систему DNS. Если вы зарегистрировали доменное имя для своего узла, то для доступа к узлу пользователям достаточно ввести его имя в своем обозревателе.

Проверка установки

Для проверки выполненной установки MIIS можно просмотреть с помощью обозревателя Internet Explorer HTML-документы в домашнем каталоге. Чтобы проверить Web-узел, подключенный к Интернету, нужно выполнить следующее:

- убедиться, что Web-сервер имеет требуемые файлы HTML в папке Wwwwroot, например создать файл home.htm, содержащий любую текстовую информацию;

- ввести адрес URL в командной строке обозревателя для файла home.htm, находящегося в домашнем каталоге нового Web-узла. Запрос на его получение нужно передавать через Web-сервер. Запустить обозреватель, например Internet Explorer, на компьютере, имеющем активное подключение к Интернету. Это может быть и проверяемый компьютер, но желательно использовать другой компьютер в сети. Адрес URL задается в командной строке обозревателя в виде строки "http://", за которой следует имя Web-узла и путь к просматриваемому файлу. Например, если узел зарегистрирован в DNS с именем "my_test.com" и нужно просмотреть файл home.htm в корневом домашнем каталоге, следует ввести строку Адрес: http://my_test.com/home.htm.

При успешной установке MIIS домашняя страница отображается в окне обозревателя.

Чтобы проверить Web-узел в интрасети, нужно выполнить следующее:

- убедиться, что компьютер имеет активное подключение к сети и работает служба сервера WINS (или другой метод определения имен);
- ввести адрес URL домашней страницы вашего нового Web-узла в командной строке обозревателя.

При успешной установке MIIS домашняя страница также отображается в окне обозревателя.

Администрирование сервера

Для администрирования MIIS используются специальные средства по настройке параметров узла, которые реализуют специальные программы-менеджеры, находящиеся в папке «Администрирование» (рис. 15.14), размещенной в папке «Панель управления».

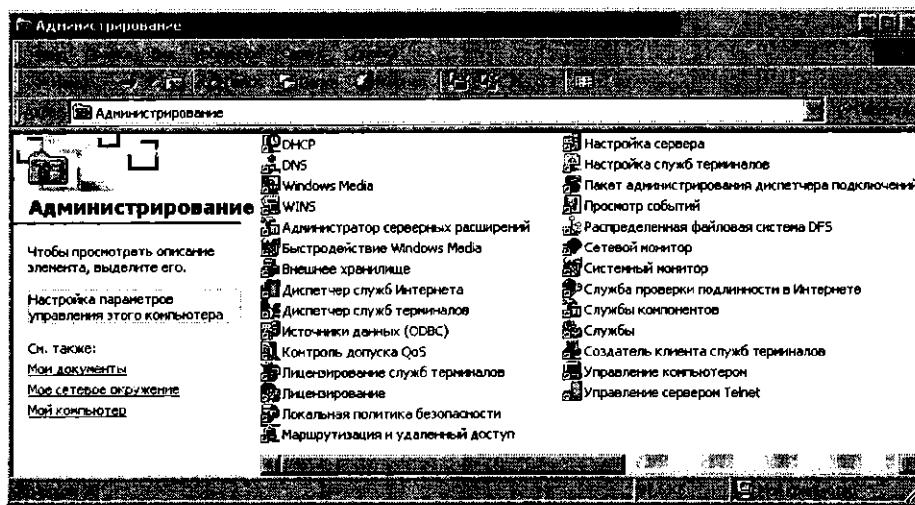


Рис. 15.14. Содержимое папки Администрирование

По сравнению с предыдущими версиями в MIIS 5.0 введены дополнительные возможности администрирования Web-узла. В общем MIIS 5.0 обеспечивает перечисляемые ниже возможности.

- Перезапуск MIIS без перезапуска компьютера.
- Создание резервной копии и восстановление MIIS с помощью резервной копии и сохранения установок метабазы для упрощения возврата в безопасное состояние.
- Предоставление сведений об использовании ресурсов процессора на сервере отдельными Web-узлами.
- Регулирование процесса выполнения внешних приложений путем ограничения доли времени, используемой для обработки внешних приложений ASP, ISAPI и CGI для отдельных Web-узлов, могут быть остановлены и перезапущены процессы, выполняющиеся неправильно.
- Гибкая реализация процесса обработки ошибок. При возникновении ошибок HTTP на Web-узлах администратор может послать специальные информативные сообщения клиентам и подробно обрабатывать ошибки ASP с помощью специального сценария ASP обработчика ошибки (обработчики ошибок помещаются в файлы 500-100.asp). Могут использоваться стандартные или пользовательские обработчики ошибок (сообщения об ошибках).
- Настройка доступа на уровне узла, виртуального каталога или файла: по «Чтению», «Записи», «Выполнению», «Использованию Сценариев».
- Удаленное управление сервером с помощью обозревателя, находящегося на удаленном компьютере любой аппаратной платформы.
- Службы терминала позволяют запускать 32-битные приложения Windows с терминалов или из эмуляторов терминалов, выполняемых на персональных компьютерах. Службы терминала позволяют практически любому компьютеру запускать приложения на сервере. Это позволяет администрировать службы Windows 2000, например MIIS, через удаленный доступ, как через консоль сервера.
- Централизованное администрирование с помощью консоли Microsoft® Management Console (MMC), которая является диспетчером служб Интернета («оснасткой»), используемой администраторами для управления своими серверами.

Рассмотрим некоторые особенности работы с диспетчером служб Интернета, который является мощным средством администрирования узлов, обеспечивающим доступ ко всем настройкам сервера. Эта программа позволяет выполнять следующее:

- управлять Web-узлом – создавать содержимое узла и формировать структуры каталогов узла;
- осуществлять администрирование сервера, для чего получать информацию о дополнительной настройке свойств узла с целью повышения про-

изводительности или безопасности, получать справочные данные администратора о параметрах реестра и метабазы.

Диспетчер служб Интернета MIIS (рис. 15.15) является инструментом администрирования MIIS 5.0, который интегрирован с другими средствами администрирования Windows 2000.

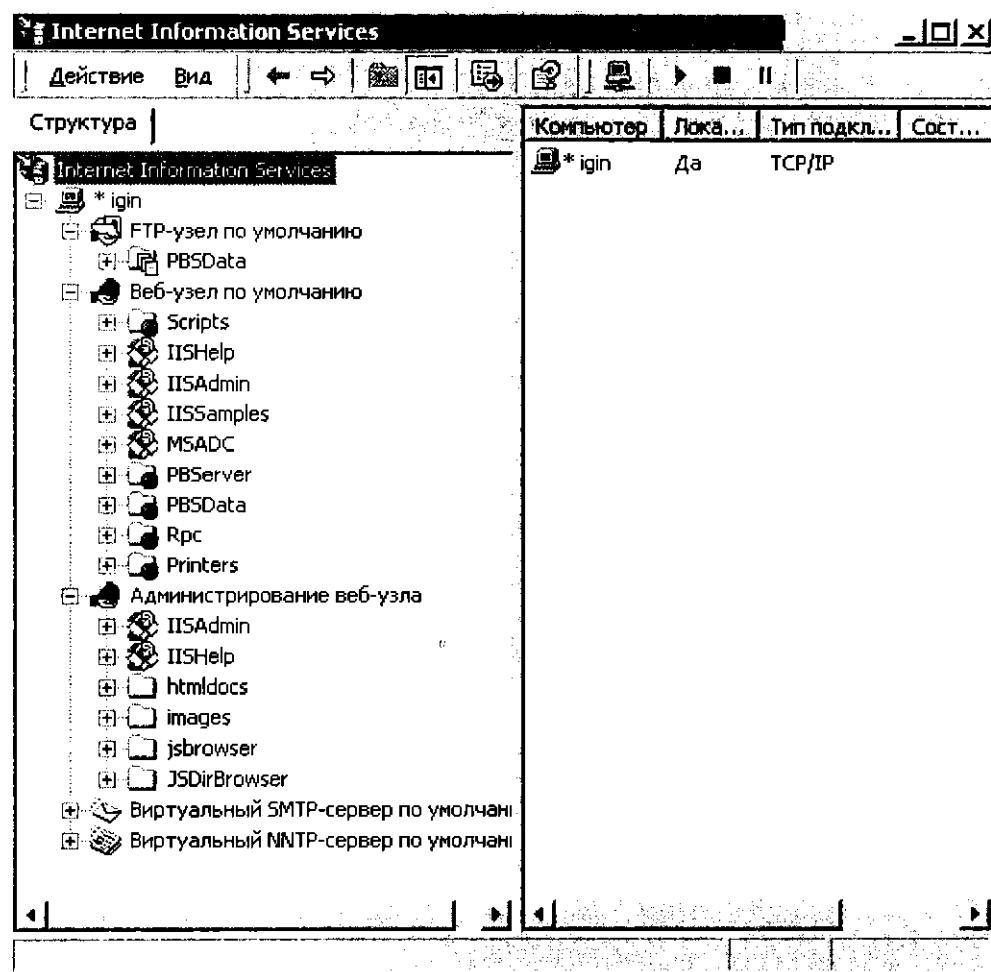


Рис. 15.15. Диспетчер служб Интернета

При установке параметров Web-узла необходимо задать, в каких каталогах будут содержаться публикуемые документы. В программе-менеджере MIIS можно указать каталоги, которые будут относиться к Web-узлу.

Можно просто скопировать файлы в домашний каталог по умолчанию C:\InetPub\WWWroot. Для узла FTP следует скопировать файлы в каталог C:\InetPub\Ftproot. Пользователи интрасети могут получить доступ к этим файлам, указав следующий адрес URL: http://ИмяСервера/ИмяФайла.

Стандартный домашний каталог создается при установке IIS и при создании нового Web-узла. Имеется возможность изменить домашний каталог. Для этого нужно открыть свойства данного Web-узла с помощью выбора пункта меню «Действие» и подпункта «Свойства» (рис. 15.16).

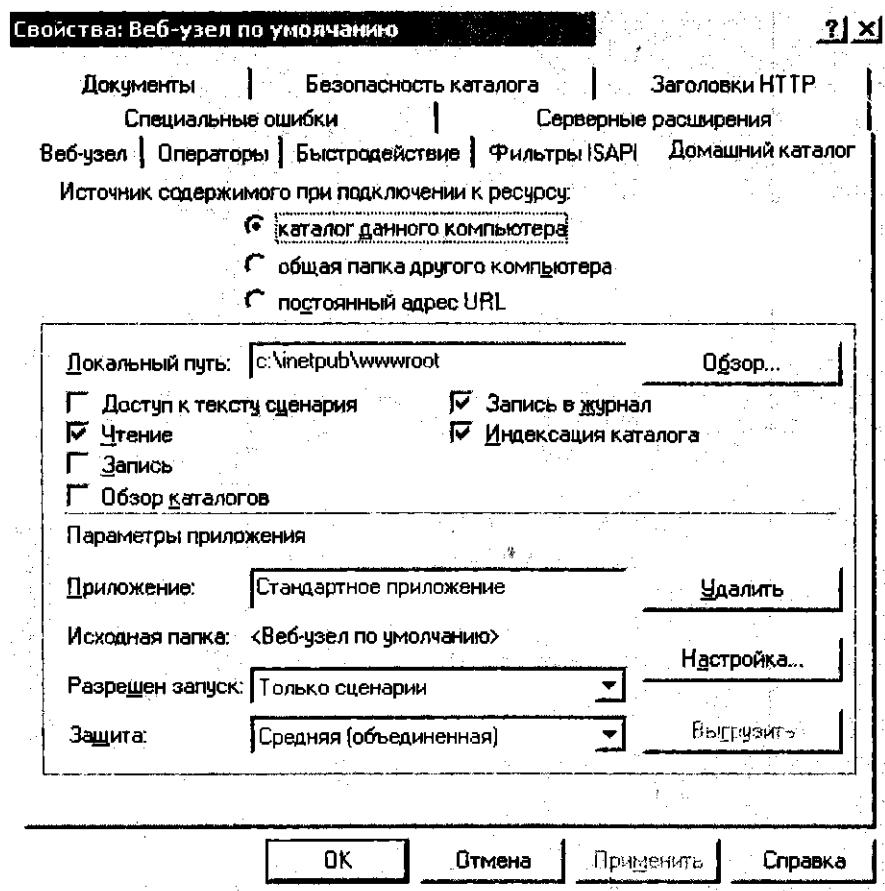


Рис. 15.16. Окно свойств Web-узла

Для публикации из любого каталога, не содержащегося в домашнем каталоге, следует создать виртуальный каталог. Виртуальный каталог имеет псевдоним (alias) — имя, которое Web-обозреватели используют для доступа к этому каталогу.

Например, если в каталоге C:\InetPub\Wwwroot\My находится файл my_home.html и для этого каталога установлен псевдоним My (имя Web-узла Server), то возможна такая форма адресации:

http://Server/My/my_home.html

Применение псевдонимов безопаснее, так как пользователи сети не имеют информации о том, где файлы физически размещаются на сервере.

Для создания виртуального каталога в программе-менеджер MIIS нужно выполнить следующее:

- выбрать Web- или FTP-узел, к которому требуется добавить каталог;
- выбрать пункт меню «Действие», подпункт «Создать» и подкоманду «Виртуальный каталог».

Для решения этой задачи целесообразно использовать Мастер создания виртуального каталога. Администрирование сервера с использованием Мастеров операционной системы Windows 2000 значительно упрощается.

Для начала отладки разработанного собственного Web-приложения при установленной службе MIIS достаточно скопировать HTML-документы в каталог C:\InetPub\Wwwroot\ (по умолчанию), а исполняемые файлы или файлы-сценариев – в каталог C:\InetPub\Wwwroot\Scripts.

15.6. Использование Apache для Microsoft Windows 9X/2000

Установка сервера

Для установки сервера Apache нужно переписать дистрибутив из сети Интернет или приобрести установочный компакт диск.

Дистрибутив сервера Apache для Windows 9X/2000 представляет собой самораспаковывающийся архив с именем “apache_1_3_14_win32.exe”, его можно получить бесплатно в сети Интернет по адресу <http://www.apache.org/>. На момент написания книги последняя версия сервера есть 1_3_14, в которой исправлены многие ошибки и внесены изменения по отзывам от пользователей предыдущих версий.

Для начала установки сервера Apache достаточно запустить файл “apache_1_3_14_win32.exe” на выполнение в среде Windows 9X/2000 и следовать указаниям программы установки.

Программа установки позволяет выбирать каталог для установки и определять состав устанавливаемых средств. Кроме того, во время установки автоматически настраиваются конфигурационные файлы, находящиеся в подкаталоге CONF. По умолчанию сервер Apache устанавливается в каталог “C:\Program Files\Apache Group\Apache”. Этот же каталог устанавливается как корневой домашний каталог Web-узла.

Запуск и управление

Для запуска и управления сервера Apache удобно использовать кнопку «Пуск», вкладку «Программы» или пункт подменю «Apache Web Server», который раскрывается в следующие пункты подменю:

- «Apache as a service» — позволяет устанавливать, деинсталлировать, запускать, перезапускать и останавливать сервер;
- «Documentation» — позволяет просматривать локально размещенную документацию и получать техническую информацию по сети Интернет;
- «Management» — позволяет просматривать конфигурационные файлы, журнал ошибок, запускать, перезапускать и останавливать сервер.

Кроме установки двоичного файла сервера разработчики Apache предусмотрели возможность перекомпиляции исходных модулей этого Web-сервера.

Компилирование сервера

Для компилирования Apache нужна среда Microsoft Visual C++ 5.0 или 6.0, причем необходимо, чтобы были установлены средства компиляции с использованием командной строки. Как установить эти средства, указано в руководстве по использованию системы программирования Microsoft Visual C++.

Перед началом компиляции нужно распаковать Apache в соответствующий каталог. Далее требуется указать подкаталог, в котором будет храниться дистрибутив Apache, в командном макросе INSTDIR. Например:

```
nmake /f Makefile.nt INSTDIR="d:\Program Files\Apache" installr
```

Подробная инструкция по компиляции находится в файле Makefile.win. Для компиляции Apache под Windows NT используют одну из следующих команд:

```
Nmake /f Makefile.win _apacher (запускают компоновку)  
Nmake /f Makefile.win _apached (отлаживают компоновку)
```

Обе команды задают компиляцию Apache, но последняя команда дополнительно вызывает создание файлов, содержащих информацию об отладке исходного кода, облегчая возможность обнаружения ошибок.

Если появляется сообщение вида «the name specified is not recognized...», то перед компиляцией нужно выполнить командный файл vcvars32.bat, введя команду:

```
"c:\Program Files\DevStudio\VC\Bin\VCVARS32.BAT "
```

где «c:\Program Files\DevStudio\VC» — каталог, в котором установлен пакет Microsoft Visual C++. Если VC++ установлен в другой каталог, то нужно указать новый путь к файлу VCVARS32.BAT.

В результате выполнения файла makefile.win будет сгенерировано ядро следующего содержания:

```
Os\win32\ApacheOS.dsp  
Regex\regex.dsp
```

```
Ap\ap.dsp
Lib\expat-lite\xmltok.dsp
Lib\expat-lite\xmlparse.dsp
Main\gen_uri_delims.dsp
Main\gen_test_char.dsp
ApacheCore.dsp
Apache.dsp
```

Кроме того, подкаталог Os\win32 содержит файлы дополнительных модулей сервера Apache.

Папка support содержит файлы для дополнительных программ, которые не выполняются непосредственно при запуске Apache, но используются администратором для возможности установки паролей и ведения log-файлов.

```
Support\htdigest.dsp
Support\htpasswd.dsp
Support\logresolve.dsp
Support\rotatelogs.dsp
```

После компиляции Apache требуется установить его в корневой домашний каталог сервера.

Чтобы автоматически установить файлы в каталог c:\ServerRoot, можно использовать команды Nmake:

```
Nmake /f Makefile.win installr INSTDIR=c:\ServerRoot (для запуска)
Nmake /f Makefile.win installd INSTDIR=c:\ServerRoot (для отладки)
```

После выполнения этих команд будут установлены следующие компоненты:

```
c:\ServerRoot\Apache.exe — файл запуска сервера
c:\ServerRoot\ApacheCore.dll — главная библиотека Apache
c:\ServerRoot\modules\ApacheModule*.dll — загружаемые модули Apache
c:\ServerRoot\bin\*.exe — программы, поддерживающие функции
администратора
c:\ServerRoot\conf — пустой каталог для файлов конфигурации
c:\ServerRoot\logs — пустой каталог для log файлов
```

Перед запуском сервера в работу требуется заполнить каталог conf. Для этого нужно скопировать файлы *.conf-dist-win из каталога conf дистрибутива в каталог c:\ServerRoot\conf и переименовать их в *.conf. Кроме того, нужно скопировать файлы conf\magic и conf\mime.types. Чтобы иметь возможность обрабатывать документы, нужно создать файл htdocs\index.html или скопировать стандартный файл Apache.

Если Apache предварительно был установлен и запущен из каталога c:\ServerRoot, то можно скопировать только те конфигурационные файлы, которые необходимо изменить в предыдущей версии сервера.

Ниже приведены команды, копирующие все файлы конфигурации в каталог c:\ServerRoot. При этом новые файлы записываются поверх старых.

```
Xcopy ..\conf\*.conf-dist-win c:\ServerRoot\conf\*.conf
Xcopy ..\conf\magic c:\ServerRoot\conf\
Xcopy ..\conf\mime.types c:\ServerRoot\conf\
Xcopy ..\htdocs\*.* c:\ServerRoot\htdocs\ /s
```

После этого в файле httpd.conf в строке @@ ServerRoot @@ нужно указать путь к корневому каталогу сервера, например “c:/ServerRoot”, используя символ “/”.

Проверка сервера

Для проверки правильности установки сервера требуется выполнить следующее:

- запустить сервер;
- запустить обозреватель;
- строке адреса набрать URL-адрес домашней страницы, например: <http://localhost/index.html>.

При правильной установке в окне обозревателя будет находиться домашняя страница (рис. 15.17).

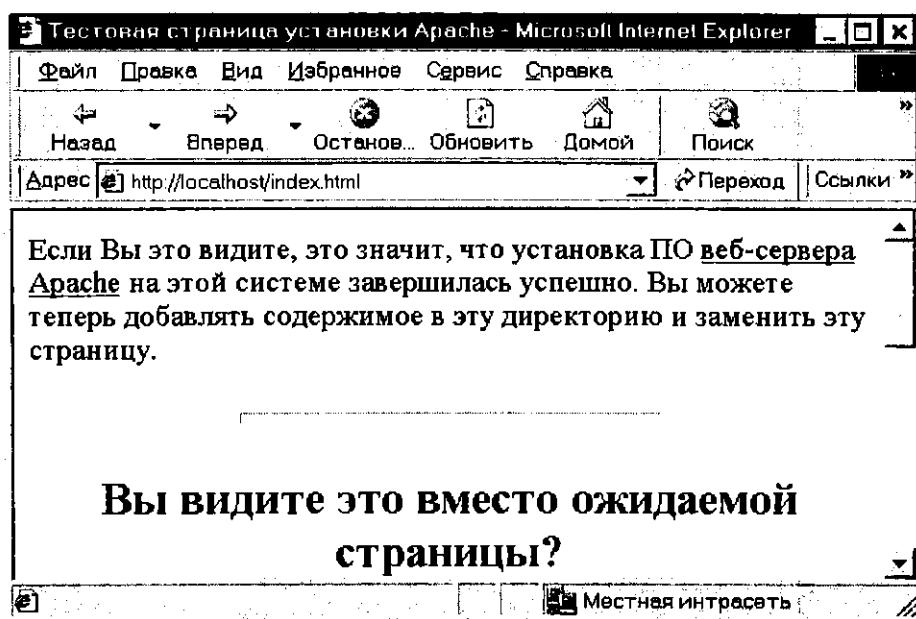


Рис. 15.17. Окно обозревателя с домашней страницей

При возникновении ошибок во время установки следует устраниить причины ошибок и повторить последовательность действий по установке сервера Apache.

Администрирование сервера

Описание всех особенностей администрирования сервера Apache выходит за рамки книги. Ниже рассматриваются основные операции, необходимые для начала работы с сервером Apache.

Сервер Apache конфигурируется с помощью файлов, находящихся в каталоге CONF. Для конфигурирования сервера Apache в основном используется файл httpd.conf, в котором находятся различные директивы администрирования. Файлы access.conf и srm.conf – это старые файлы, не используемые большинством администраторов, но присутствующие в установке для совместимости с предыдущими версиями сервера (в них по умолчанию директивы отсутствуют).

В файле **Httpd.conf** содержится вспомогательная информация, являющаяся комментариями (комментарии начинаются с символа "#"), остальные строки являются директивами сервера Apache.

Директивы администрирования сервера Apache сгруппированы в три основные секции:

- управления сервером Apache в целом (управления глобальными переменными окружения сервера);
- определяющие параметры сервера по умолчанию, которые относятся ко всем виртуальным хостам;
- отвечающие за установки параметров для конкретных виртуальных хостов.

Рассмотрим особенности использования наиболее важных директив файла **Httpd.conf** для первоначального использования сервера.

Директива **ServerRoot** используется для установки корневого домашнего каталога для Web-сервера и имеет следующий формат:

ServerRoot directory-filename

Например, директива

ServerRoot "C:/Program Files/Apache Group/Apache"

устанавливает каталог "C:/Program Files/Apache Group/Apache" в качестве корневого домашнего каталога.

Директива **Alias** используется для создания псевдонима (алиаса) виртуального каталога Web-узла и имеет следующий формат:

Alias url-path directory-filename

где: **url-path** – имя (алиас) виртуального каталога; **directory-filename** – относительный путь к соответствующему локальному каталогу.

Псевдонимы позволяют хранить файлы Web-узла не только в корневом домашнем каталоге. Например, если использовать директиву вида

```
Alias /my /ftp/pub/image
```

то для обращения к файлу `/ftp/pub/image/index.html` можно использовать следующий запрос:

```
http://myserver/my/index.html
```

Директива `ScriptAlias` используется с целью создания псевдонима виртуального каталога Web-узла для хранения сценариев и имеет такой же формат записи, как у предыдущей директивы. Например, после задания директивы

```
ScriptAlias /cgi-bin/ /web/cgi-bin/
```

для обращения к сценарию `/web/cgi-bin/foo` можно использовать следующий запрос:

```
http://myserver/cgi-bin/my.
```

Отметим, что при установке сервера Apache рассмотренные директивы используются в файле `Httpd.conf` с параметрами по умолчанию. Вместе с файлом `Httpd.conf` находится файл `Httpd.conf.default`, в котором хранится первоначальное содержимое файла `Httpd.conf`, используемое для установки параметров сервера по умолчанию.

15.7. Варианты создания Web-узла

Для тех, кто никогда раньше не занимался установкой и работой с Web-узлом, существует возможность открыть свой собственный сайт на узле провайдера или организовать управление своим узлом через удаленный сервер, принадлежащий провайдеру. Такие варианты организации собственного узла имеют ряд достоинств.

Во-первых, организация, управление и администрирование такого Web-узла не требует больших знаний и опыта. Решением этих вопросов будет заниматься профессиональный персонал провайдера, с помощью которого можно качественно оформить узел, выбрать дизайнерский стиль, наделить сайт набором сетевых услуг. В перечень этих услуг входит сбор статистики по числу обращений к узлу, числу повторяющихся обращений, продолжительности времени, проведенного пользователями на определенных Web-страницах, адреса пользователей, ведение регистрационных журналов ошибок, сообщений, и публикации отчетов, использование БД для хранения различной структурированной информации. При сотрудничестве с опытным персоналом узла провайдера можно быстро приобрести опыт, изучать и сравнивать различные услуги и типы серверных сред.

Во-вторых, потребуется гораздо меньше денежных средств и времени на организацию и содержание такого узла, чем на установку и поддержку собственного Web-сервера.

Подключение организации к Интернету для публикации собственной информации представляет собой важную и, бесспорно, не простую задачу. Перед тем как приступить к установке своего Web-сервера, желательно изучить как можно больше чужой опыта во избежание лишней траты времени и денег.

Таким образом, для организации собственного Web-узла можно реализовать следующие варианты:

- открыть свой собственный сайт на узле надежного провайдера;
- организовать управление своим узлом через удаленный сервер, принадлежащий провайдеру;
- подготовить или нанять специалистов и разработать Web-узел собственными силами.

Вариантов построения Web-узла существует очень много. Прежде чем приступить к выбору варианта разработки собственного Web-узла, следует разобраться с технологиями в области разработки Web-приложений, лежащих в основе любого Web-узла, и в том числе с публикацией БД на страницах Web-узла — принципиально важной технологией, применяемой в большинстве Web-узлов сети Интернет и особенно сетей интранет.

Контрольные вопросы и задания

1. Приведите упрощенную схему функционирования Web-приложения.
2. Назовите принципы построения сетей интранет.
3. Изобразите архитектуру Web-приложения с модулями расширения сервера.
4. Охарактеризуйте интерфейсы CGI и WinCGI.
5. Какие способы могут использоваться для запуска модуля расширения сервера?
6. Охарактеризуйте интерфейсы ISAPI/NSAPI.
7. Приведите схему Web-приложения с использованием модулей расширения обозревателя.
8. Охарактеризуйте средства, используемые для разработки модулей расширения клиентской части.
9. Изобразите архитектуру и опишите схему функционирования двухуровневого Web-приложения, использующего БД.
10. Охарактеризуйте архитектуру трехуровневого Web-приложения, использующего БД.
11. Изобразите архитектуру и опишите схему функционирования многоуровневого Web-приложения с сервером приложений.
12. Приведите архитектуру многоуровневого смешанного Web-приложения.
13. Охарактеризуйте архитектуру многоуровневого Web-приложения на основе технологии CORBA.

14. Дайте общую характеристику интерфейсам OLE DB, ADO и ODBC.
15. Что представляет собой Web-сервер и для каких целей он используется?
16. Дайте краткую характеристику основным операционным системам, используемым в качестве платформ Web-серверов.
17. Охарактеризуйте сервер Apache и укажите поддерживаемые им функции.
18. Назовите достоинства и недостатки сервера Internet Information Server в сравнении с сервером Apache.
19. Какие возможности обеспечивает сервер Microsoft Personal Web Server?
20. Каким образом осуществляется администрирование серверами Personal Web Server и Internet Information Server?
21. С помощью какой среды и как выполняется компилирование исходных модулей сервера Apache?
22. Назовите возможные варианты создания Web-узла.

Литература

1. Вебер Дж. Технология JavaФ в подлиннике / Пер. с англ. – СПб.: БНВ-Санкт-Петербург, 1999.
2. Дунаев С.Б. Технологии Интернет-программирования. – СПб.: БХВ-Петербург, 2001.
3. Коннолли Т., Бэгг К. Базы данных. Проектирование. Реализация и сопровождение. Теория и практика. – М.: Вильямс, 2003.
4. Мещеряков Е.В., Хомоненко А.Д. Публикация баз данных в Интернете. – СПб.: БХВ-Петербург, 2001.
5. Орфали Р., Харки Д. Java и CORBA в приложениях клиент-сервер / Пер. с англ. – М.: ЛОРИ, 2000.
6. Фролов А.В., Фролов Г.В. Базы данных в Интернете: практическое руководство по созданию Web-приложений с базами данных. – М.: Издательско-торговый дом «Русская редакция», 2000.

16. Интерфейсы программирования Web-приложений

В главе рассматриваются особенности создания Web-приложений с применением модулей расширения Web-сервера, используемых для публикации БД на HTML-страницах на основе интерфейсов программирования серверных расширений CGI и ISAPI. Интерфейс CGI продолжают широко использовать в UNIX-подобных системах благодаря быстроте разработки и надежности функционирования Web-приложений, использующих этот интерфейс. Интерфейс WinCGI редко используется для разработки Web-приложений, так как он предназначен для серверов, функционирующих в Windows-подобных операционных системах, в которых предпочтительно использовать интерфейс ISAPI. Поэтому интерфейс WinCGI нами не рассматривается.

Интерфейсы программирования расширений сервера предоставляют более широкий спектр возможностей по сравнению с технологиями ASP, IDC/HTX, поэтому разработанные на их основе модули можно использовать для реализации нестандартных функций.

Например, эти модули могут использоваться обеспечения дополнительных мер по обеспечению безопасности передачи данных по линиям связи (создание ISAPI-фильтров), для организации доступа к данным через нестандартные интерфейсы или внешние устройства (например, в случае создания Интернет-магазина при передаче данных о платежах клиентов) и т. д. В главе освещены также основные интерфейсы ODBC, OLE DB и ADO, используемые в модулях расширения Web-сервера для доступа к БД.

Приводятся примеры, реализованные в среде системы программирования Visual C++6.0. При подготовке материала главы предполагалось, что читатель знаком с основами программирования на языке C++.

16.1. Общий интерфейс взаимодействия CGI

CGI-модуль представляет собой консольный exe-файл, загружаемый сервером для обработки данных обозревателя и генерации HTML-документа. Для обмена данными между сервером и CGI-модулем используются стандартные входной и выходной потоки. Кроме того, для передачи параметров и получения системной информации могут использоваться переменные окружения. При использовании CGI-интерфейса имеется возможность динамически (в ответ на запрос пользователя из обозревателя) формировать HTML-документ.

CGI-модуль может вызываться из обозревателя для динамического формирования HTML-документов на основании данных, введенных посетителями сервера Web при помощи форм или путем выбора ими ссылок. Напомним, что обработка данных, введенных пользователем в интерфейсные элементы формы, может быть выполнена с помощью обработчика, URL которого задается параметром ACTION тега формы, а способ передачи данных из формы на Web-сервер определяет параметр METHOD тега формы.

Напомним, что параметр METHOD позволяет выбрать метод передачи данных Web-серверу. При разработке CGI-модуля необходимо учитывать, каким методом HTTP-протокола будут передаваться данные между обозревателем и сервером.

Основными методами передачи параметров запроса являются методы POST и GET. Метод POST применяется для передачи информации, включенной в HTTP-запрос и относящейся к ресурсу, указанному URL-адресом CGI-модуля. Метод GET используется для извлечения информации, передаваемой в URL-адресе CGI-модуля. В зависимости от используемого метода в HTTP-запросе сервер формирует передаваемые для CGI-модуля параметры различными способами. В случае применения метода GET для получения параметров запроса используются переменные окружения (переменные операционной системы MS-DOS, устанавливаемые командой SET), в случае применения метода POST данные передаются через стандартный поток ввода (рис. 16.1).

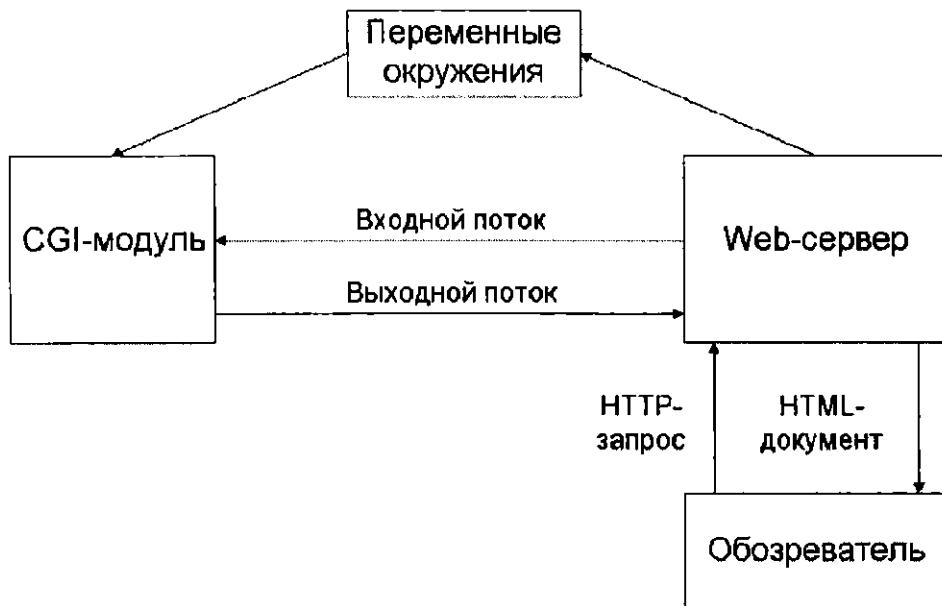


Рис. 16.1. Схема взаимодействия сервера и CGI-модуля

Например:

```
<HTML><HEAD><TITLE>Пример CGI-модуля</TITLE>
</HEAD>
<BODY>
<A href="http://localhost/scripts/my_cgi.exe?TEST=modul">
Загрузить CGI-модуль по запросу http://localhost/Scripts/
my_cgi.exe?TEST=modul</A>
<FORM action=http://localhost/scripts/my_cgi.exe method=post>
<B>Введите ваше имя:</B>
<INPUT maxLength=60 name=NAME size=40 value="My_login">
<BR>Тип действия: <INPUT CHECKED name=TYPE type=radio
value=Registry> Зарегистрировать
<INPUT name=TYPE type=radio value=Execute>Выполнить сценарий
<BR><INPUT name=Submit type=submit value=Send>
<BR><INPUT name=Reset type=reset value=Сброс><BR>
</FORM></P></BODY></HTML>
```

В этом примере в окне обозревателя будут отображены ссылка — Загрузить CGI-модуль по запросу `http://localhost/Scripts/my_cgi.exe?TEST=modul` и элементы управления, входящие в состав формы. По нажатию на специальную кнопку “Послать запрос”, имеющую тип `Submit`, обозреватель посылает запрос Web-серверу. Последний, в свою очередь, запускает CGI-модуль, находящийся по URL-адресу `http://localhost/scripts/my_cgi.exe`. При этом данные, введенные пользователем в интерфейсные элементы формы, передаются через стандартный поток ввода. При выборе пользователем ссылки “Загрузить CGI-модуль по запросу `http://localhost/Scripts/my_cgi.exe?TEST=modul`” обозреватель передает запрос Web-серверу на загрузку CGI-модуля, находящегося по URL-адресу `http://localhost/Scripts/my_cgi.exe`. При этом для передачи запроса используется метод `GET` и сервер передает CGI-модулю только данные, находящиеся в командной строке — “`TEST=modul`” через переменную окружения `QUERY_STRING`.

Метод `GET` может использоваться и при передачи данных формы. В этом случае в теге формы указывается `METHOD=GET`. При использовании метода `GET` данные находятся в переменной окружения `QUERY_STRING` в следующем формате:

“Имя1=Значение1&Имя2=Значение2&Имя3=Значение3”

где Имя1,...,Имя3 – значения параметров `NAME`, задающих имена полей формы. При этом количество параметров определяется количеством элементов формы (в данном случае – три). Вместо значений параметров подставляются данные из соответствующих полей.

Отметим, что значения не всех интерфейсных элементов формы передаются на сервер. Например, значение служебного элемента типа **reset**

```
<INPUT name=Reset type=reset value=Сброс><BR>
```

не передается на сервер, так как оно не несет полезной информации.

Если в приведенном выше примере HTML-документа параметру **METHOD** задать значение **GET**, то переменная **QUERY_STRING** примет значение

```
NAME=My_login&TYPE=Registry&Submit=Send
```

При этом значением переменной **QUERY_STRING** является закодированная строка. В этой строке символы пробелов заменяются на символы “+”, а при использовании специальных и управляющих символов производится их замена на последовательность символов вида “Ххх”, где “хх” — шестнадцатеричный код символа.

Если в названии или в полях интерфейсных элементов использовать символы, набранные с использованием альтернативной кодировки, то при их передаче сервер будет также заменять каждый такой символ шестнадцатеричным кодом, перед которым будет находиться символ “%”.

Например, если первый интерфейсный элемент заменить на следующий **<INPUT maxLength=60 name=ИМЯ size=40 value="МОЕ ИМЯ">**, то переменная **QUERY_STRING** будет иметь следующее значение

```
%C8%CC%DF=%CC%CE%C5+%C8%CC%DF&TYPE=Registry&Submit=Send
```

В результате работы CGI-модуль генерирует HTML-документ, который помещается в стандартный поток вывода. Web-сервер использует для управления и передачи различных данных CGI-модулю много переменных окружения. Рассмотрим особенности использования переменных окружения и стандартных потоков ввода/вывода.

Переменные окружения

Переменные окружения используются в CGI-модуле для получения от Web-сервера служебной информации о программном обеспечении Web-сервера, параметрах HTTP-запроса и другой информации, передаваемой Web-сервером CGI-модулю. Например, в случае использования метода **GET** при формировании HTTP-запроса в переменной **QUERY_STRING** находятся передаваемые пользовательские данные, а при использовании метода **POST** в переменных окружения **CONTENT_TYPE** и **CONTENT_LENGTH** содержатся тип и длина передаваемой информации соответственно, сами данные в этом случае передаются через стандартный входной поток.

В переменных окружения содержатся следующие данные:

- **SERVER_NAME** – символьическое имя или IP-адрес компьютера, на котором запущен Web-сервер (задается в URL при обращении к этому Web-серверу);

- **SERVER_SOFTWARE** – название и версия Web-сервера, разделенные символом “/”;
- **GATEWAY_INTERFACE** – версия CGI-интерфейса;
- **SERVER_PROTOCOL** – название и версия протокола передачи данных, используемого на сервере, разделенные символом “/”;
- **SERVER_PORT** – номер порта, на который обозреватель посыпает запросы Web-серверу;
- **REQUEST_METHOD** – метод HTTP-запроса;
- **CONTENT_LENGTH** – число символов в стандартном входном потоке;
- **CONTENT_TYPE** – тип данных, находящихся в стандартном входном потоке;
- **SCRIPT_NAME** – виртуальный путь к исполняемому CGI-модулю, используемый для получения URL в CGI-модуле;
- **PATH_INFO** – полученный от клиента виртуальный путь к CGI-модулю;
- **PATH_TRANSLATED** – физический путь до CGI-модуля, преобразованный из значения **PATH_INFO**;
- **QUERY_STRING** – строка символов, следующая за знаком “?” в URL данного запроса;
- **REMOTE_HOST** – символическое имя удаленной машины, с которой произведен запрос;
- **REMOTE_ADDRESS** – IP-адрес клиента;
- **AUTH_TYPE** – метод аутентификации (подтверждения подлинности), если Web-сервер поддерживает аутентификацию пользователей и CGI-модуль защищен от постороннего доступа;
- **REMOTE_USER** – имя пользователя в случае аутентификации;
- **REMOTE_IDENT** – имя пользователя, полученное от сервера (если сервер поддерживает аутентификацию);
- **HTTP_ACCEPT** – список типов MIME, известных клиенту и отделенных друг от друга запятой (тип/подтип, тип/подтип и т. д.);
- **HTTP_USER_AGENT** – имя обозревателя, с которого послан запрос;
- **HTTP_REFERER** – адрес URL документа HTML, из которого осуществляется вызов CGI-модуля;
- **HTTP_ACCEPT** – типы данных MIME, которые могут быть приняты обозревателем от Web-сервера;
- **HTTP_ACCEPT_LANGUAGE** – идентификатор национального языка для получения ответа от Web-сервера;
- **HTTP_UA_PIXELS** – разрешение видеoadаптера, установленное в компьютере пользователя;
- **HTTP_UA_COLOR** – допустимое число цветов в системе пользователя;
- **HTTP_UA_CPU** – тип центрального процессора в компьютере пользователя;

- **HTTP_UA_OS** – операционная система, под управлением которой работает обозреватель;
- **HTTP_CONNECTION** – тип соединения;
- **HTTP_HOST** – имя узла, на котором работает Web-сервер;
- **HTTP_ACCEPT_ENCODING** – тип метода кодирования, используемый обозревателем для формирования запроса Web-серверу;
- **HTTP_FROM** – имя пользователя, установленное в настройках обозревателя;
- **HTTP_PRAGHA** – специальные команды Web-серверу;
- **HTTP_AUTHORIZATION** – информация для аутентификации обозревателя на Web-сервере.

Замечание.

Тип MIME (Multipurpose Internet Mail Extensions – Многоцелевые расширения почтового стандарта Интернета) определяет протокол передачи почтовых сообщений, используемый взамен стандартного.

Пример 1. Для Web-сервера Microsoft-PWS-95/2.0 в среде Windows 98 при условии размещения модуля CGI с именем **my_cgi.exe** на сервере **igin** переменные окружения могут иметь следующие значения:

```
SERVER_SOFTWARE = Microsoft-PWS-95/2.0
SERVER_NAME = igin
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
REQUEST_METHOD = POST
HTTP_ACCEPT = image/gif, image/x-bitmap, image/jpeg, image/pjpeg,
image/png, /*
PATH_TRANSLATED = C:\WebShare\wwwroot\
SCRIPT_NAME = /scripts/my_cgi.exe
QUERY_STRING = TEST=modul
REMOTE_HOST = 127.0.0.1
REMOTE_ADDR = 127.0.0.1
CONTENT_TYPE = application/x-www-form-urlencoded
HTTP_ACCEPT = image/gif, image/x-bitmap, image/jpeg, image/pjpeg,
image/png, /*
HTTP_USER_AGENT = Mozilla/4.7 [en] (Win98; I)
HTTP_ACCEPT_LANGUAGE = en,ru
HTTP_CONNECTION = Keep-Alive
HTTP_HOST = igin
HTTP_ACCEPT_ENCODING = gzip
```

Остальные переменные имеют значение null.

Здесь: **igin** – имя компьютера, где запускается CGI-модуль;

C:\WebShare\wwwroot\ – каталог где находится CGI-модуль; my_cgi.exe – имя файла, содержащего модуль.

Пример 2. Для Web-сервера Apache/1.3.14 (содержащего модуль CGI в папке cgi-bin сервера igin) следующие переменные окружения могут иметь значения, отличные от приведенных ранее:

```
SERVER_SOFTWARE = Apache/1.3.14 (Win32)
REMOTE_HOST = (null)
HTTP_HOST = igin
PATH_TRANSLATED = d:\program\apache\htdocs\
SCRIPT_NAME = /cgi-bin/my_cgi.exe
```

Пример 3. Для Web-сервера Microsoft-IIS/5.0 в среде Windows 2000 Server (содержащего модуль CGI в папке scripts) следующие переменные окружения могут иметь значения, отличные от приведенных выше:

```
SERVER_SOFTWARE = Microsoft-IIS/5.0
SERVER_NAME = igin
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.1
PATH_TRANSLATED = c:\inetpub\wwwroot
SCRIPT_NAME = /Scripts/my_cgi.exe
REMOTE_HOST = 127.0.0.1
REMOTE_ADDR = 127.0.0.1
```

Для вывода значения этих переменных может использоваться следующий код на языке C++:

```
#include <stdio.h>
#include <stdlib.h>
#define N 31
main(int argc, char *argv[])
{
    char *par[]={“SERVER_SOFTWARE”,
    “SERVER_NAME”,
    ...
    “HTTP_AUTHORIZATION”
    };
    for(x=0;x<N;x++)
        printf(“%s = %s<br>”,par[x],getenv(par[x]));
}
```

В приведенном коде вместо “...” нужно подставить названия требуемых переменных окружения.

Стандартный вывод

После перекодировки входных данных в CGI-модуле необходимо сформировать HTML-документ в стандартном устройстве вывода. Причем, HTML-документы могут создаваться динамически с использованием информации из БД. При этом HTML-страница формируется на основании информации, хранимой в БД, которая может обновляться в реальном масштабе времени.

Например, следующий фрагмент кода на C++ позволяет выводить в стандартный поток вывода stdout простейший HTML-документ:

```
printf("Content-type: text/html<BR>");  
printf("<HTML><HEAD> <TITLE> Простейший HTML-документ  
</TITLE></HEAD>");  
printf("<BODY >");  
printf("<H1>Результаты обработки входных данных формы</H1>");
```

При этом первая строка "Content-type: text/html" есть директива CGI-интерфейса, которая определяет MIME-тип возвращаемого документа (в нашем случае – документ в формате HTML). В следующем разделе рассмотрен пример, демонстрирующий возможности интерфейса CGI по динамическому формированию HTML-документа.

Пример CGI-модуля на Visual C++

Рассмотрим пример CGI-модуля, позволяющий перекодировать передаваемые данные при использовании метода GET и POST.

Для получения данных, относящихся к требуемому значению параметра из текстовой строки, содержащейся в переменной окружения QUERY_STRING или в стандартном потоке ввода, может использоваться следующий код на C++:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#pragma hdrstop  
//-----  
//-----  
#define MAX_ARRAY 40  
#define MAX_LENGTH_BUFFER 10000  
//Структура, используемая для хранения передаваемых параметров  
typedef struct {  
    char name[50];  
    char value[100];  
} param;
```

```
main(int argc, char *argv[])
{
    param arr[MAX_ARRAY];
    char *def ;
    int m=0;
    int x,y=0;
    printf("Content-type: text/html%c%c",10,10);
    //Следующий код используется для определения метода
    //передачи данных и для считывания данных в буфер
    if(strcmp(getenv("REQUEST_METHOD"),"POST"))
        //функция getenv возвращает значение переменной окружения,
        //название которой задано в качестве аргумента
    {
        if(strcmp(getenv("REQUEST_METHOD"),"GET"))
        {
            printf("Внутренняя ошибка с кодом 1");
            exit(1);
        }
        printf("Используется метод GET<BR>");
        // Считываем передаваемые данные в буфер def из
        // переменной окружения QUERY_STRING
        def=getenv("QUERY_STRING");
    }
    else{
        printf("Используется метод POST<BR>");
        int leng = atoi(getenv("CONTENT_LENGTH"));
        // Считаем передаваемые данные в буфер def из
        // стандартного потока ввода STDIN
        def = (char *) malloc(MAX_LENGTH_BUFFER);
        fread(def, leng, 1, stdin);
    }
    int len=strlen(def);
    if (len==0)
    {
        printf("Внутренняя ошибка с кодом 4");
        exit(1);
    }
    //Промежуточный буфер word используется при декодировании данных
    char *word = (char *) malloc(MAX_LENGTH_BUFFER);
    //Цикл для декодирования данных
    for(x=0;x<len;x++,y++)
    {
        if(def[x] == '%') {
```

```
register char d;
//Код для декодирования шестнадцатеричного кода символов
d=(def[x+1] >= 'A' ? ((def[x+1] & 0xdf) - 'A')+10 : (def[x+1] - '0'));
d*=16;
d+=(def[x+2] >= 'A' ? ((def[x+2] & 0xdf) - 'A')+10 : (def[x+2]- '0'));
x+=2;
word[y]=d;
continue;}
else
//Замена символа '+' на пробел
if(def[x] == '+') {word[y] = ' ';continue;}
else
if(def[x] == '=') {
word[y]='\0';
y=-1;
//формирование структуры для хранения передаваемых параметров
strcpy(arr[m].name,word);
continue;}
else
if(def[x] == '&') {
word[y]='\0';
//формирование структуры для хранения передаваемых параметров
strcpy(arr[m].value,word);
m++;
y=-1;
continue;}
else
word[y]=def[x];
}
word[y]='\0';
//формирование структуры для хранения передаваемых параметров
strcpy(arr[m].value,word);
m++;
printf("Значения передаваемых параметров:");
for(x=0; x < m; x++)
{
//вывод передаваемых параметров
printf("<li>%s = %s</li>",arr[x].name, arr[x].value);
}
return 0;
}
```

В приведенном примере используется функция `getenv()`, которая возвращает значения соответствующей переменной окружения.

В начале работы программы производится анализ используемого метода для передачи данных. Если используется метод GET, то для доступа к передаваемым данным используется следующий код

```
def=getenv("QUERY_STRING");
```

В случае использования метода PUT в коде применяется функция `fread()`:

```
def = (char *) malloc(MAX_LENGTH_BUFFER);
int leng = atoi(getenv("CONTENT_LENGTH"));
fread(def, leng, 1, stdin);
```

Отметим, что при использовании метода POST CGI-модуль получает данные из формы через стандартный поток ввода `stdin`, в котором данные находятся в том же формате, что и в случае метода GET.

Использование метода POST предпочтительно по сравнению с методом GET для передачи больших по размерам данных. Учитывая, что операционные системы, как правило, накладывают ограничения на размер данных, передаваемых через переменную окружения `QUERY_STRING`, метод GET обычно применяют для обработки данных, передаваемых через URL, или данных форм, имеющих небольшое число интерфейсных элементов.

Количество байтов данных, которые нужно считать из стандартного потока ввода, считывается с помощью следующего кода

```
int leng = atoi(getenv("CONTENT_LENGTH"));
```

Здесь переменная окружения с именем `CONTENT_LENGTH` содержит длину буфера данных, находящихся в потоке `stdin`.

Строка

```
def = (char *) malloc(MAX_LENGTH_BUFFER);
```

динамически выделяет `MAX_LENGTH_BUFFER` байтов под буфер для хранения передаваемых данных.

Для вызова приведенного CGI-модуля нужно откомпилировать код модуля в системе программирования VisualC++6.0 или C++BUILDER5.0 под именем `my_cgi.exe` и поместить его в каталог `C:\WebShare\Scripts\` в случае запуска в среде Windows9X под управлением Personal Web Server. В случае запуска в среде Windows2000 под управлением Microsoft Internet Information Server в код модуля нужно поместить в каталог `c:\inetpub\scripts\` и в случае запуска в любой среде под управлением сервера Apache — поместить в каталог `d:\program\apache\cgi-bin\` (каталог `d:\program\apache\` — это каталог, в который установлен сервер Apache).

При запуске CGI-модуля из HTML-документа в случае серверов PSW и MIIS может использоваться следующий URL – http://localhost/scripts/my_cgi.exe, а в случае сервера APACHE – http://localhost/cgi-bin/my_cgi.exe.

Кроме того, для вызова CGI-модуля необходим HTML-документ. Например, если загрузить приведенную выше форму в окне обозревателя и выбрать следующую ссылку

Загрузить CGI-модуль по запросу
http://localhost/Scripts/my_cgi.exe?TEST=modul

то будет получен следующий текст в окне обозревателя:

Используется метод GET

Значения передаваемых параметров:

TEST = modul

При нажатии на кнопку Send будет выведена следующая информация:

Используется метод POST

Значения передаваемых параметров:

NAME = My_login

TYPE = Registry

Submit = Send

Перейдем к рассмотрению технологий публикации БД из CGI-модуля на HTML-документ.

Публикация БД с использованием ODBC

Для публикации БД из CGI-модуля на HTML-документ могут применяться различные интерфейсы доступа к БД – ODBC, ADO и др. В этом разделе рассмотрим основные особенности использования в CGI-модуле традиционного интерфейса ODBC и в следующем разделе – перспективного интерфейса ADO.

Интерфейс ODBC может использоваться при создании Web-приложений для связи модулей расширений Web-сервера (CGI, WinCGI и ISAPI) с базами данных и при применении стандартных средств, входящих в состав Web-серверов, таких как ASP и IDC/HTX-страницы. На стороне обозревателя могут также организовываться обращения к интерфейсу ODBC из объектов ActiveX и Java-апплетов.

ODBC представляет собой программный интерфейс, разработанный фирмой Microsoft для доступа к реляционным БД. Он организован в виде библиотеки функций, используемых для доступа к базам данных на языке SQL.

Этот интерфейс не является объектным и поэтому, например в серверных страницах ASP, получить доступ к источникам данных с использованием ODBC можно только через интерфейс ADO. В связи с этим интерфейс ODBC

недоступен из серверных сценариев JScript и VB Script, расположенных в страницах ASP. Но в традиционной технологии IDC/HTX-страниц возможно использование интерфейса ODBC с помощью объектов IDC/HTX.

Использование этого интерфейса предполагает установку драйвера ODBC для конкретного типа СУБД. Например, драйвер ODBC стандартно входит в состав СУБД, созданных фирмой Microsoft. Драйверы ODBC входят в состав многих СУБД, рассчитанных на работу в среде операционных систем Microsoft Windows, Macintosh и некоторых версий Unix.

Кроме того, дополнительно в состав операционной системы включается администратор ODBC, позволяющий устанавливать параметры соединения с источником данных.

Для обеспечения доступа к конкретному источнику данных используются системные имена источника данных DSN (Data System Name). Например, в операционных системах Microsoft Windows администратор ODBC входит в состав утилит панели управления и имеет название «Администратор источников данных ODBC» (рис. 16.2).

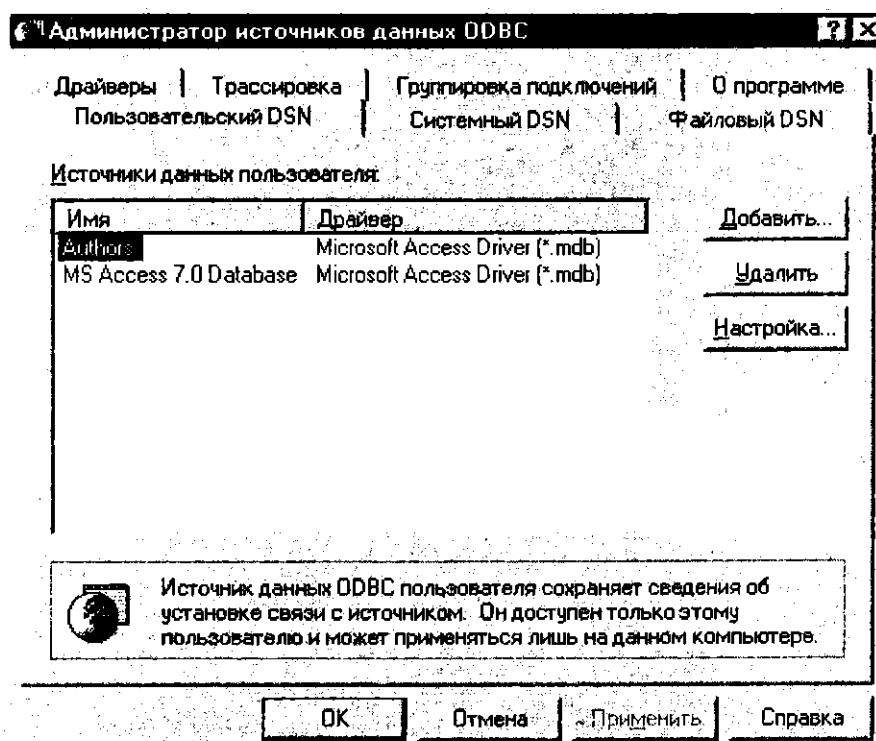


Рис. 16.2. Окно Администратора источника данных ODBC

Администратор ODBC позволяет вводить новые пользовательские DSN и перенастраивать существующие имена. При настройке имен DSN имеется возможность связывать каждое имя DSN с конкретной БД.

При использовании интерфейса ODBC для доступа к БД применяются функции, входящие в состав библиотеки ODBC, а запросы для извлечения требуемых данных составляются на языке SQL.

Пример 1. Создание CGI-модуля — приложения “OD.EXE”, раскрывающего основные возможности интерфейса ODBC для создания простейших Web-приложений, связанные с доступом к информации в БД с помощью языка SQL.

```
//Стандартные заголовочные файлы
#include <windows.h>
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>
#include <odbcss.h>
#include <string>
using namespace std;
//Заголовочный файл, генерируемый средой Visual C++
#include "stdafx.h"

int main(int argc, char* argv[])
{
    //Определение системных переменных
    SQLHENV sql_henv = SQL_NULL_HENV;
    SQLHDBC sql_hdbc = SQL_NULL_HDBC;
    SQLHSTMT sql_hstmt = SQL_NULL_HSTMT;
    RETCODE err;
    UCHAR sys_DSN[SQL_MAX_DSN_LENGTH + 1] = "Authors";
    UCHAR user_Name[MAXNAME] = "";
    UCHAR user_Password[MAXNAME] = "";
    //Инициализация среды выполнения приложения
    err = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE,
    &sql_henv);
    //Анализ кода ошибки
    if(err != SQL_SUCCESS && err != SQL_SUCCESS_WITH_INFO) return 1;
    //установка параметров среды выполнения приложения
    err = SQLSetEnvAttr(sql_henv, SQL_ATTR_ODBC_VERSION,
    (SQLPOINTER)SQL_OV_ODBC3, SQL_IS_INTEGER);
    //Анализ кода ошибки
    if(err != SQL_SUCCESS && err != SQL_SUCCESS_WITH_INFO)
    return 1;
```

```
//инициализация среды соединения
err = SQLAllocHandle(SQL_HANDLE_DBC, sql_henv , &sql_hdbc );
if(err != SQL_SUCCESS && err != SQL_SUCCESS_WITH_INFO)
return 1;
//установка соединения
err = SQLConnect(sql_hdbc,
sys_DSN, (SWORD)strlen((const char*)sys_DSN),
user_Name, (SWORD)strlen((const char*)user_Name),
user_Password, (SWORD)strlen((const char*)user_Password));
if(err != SQL_SUCCESS && err != SQL_SUCCESS_WITH_INFO)
{
printf("SQLConnect error\n");
return 1;
}
// получение идентификатора команд
err = SQLAllocHandle(SQL_HANDLE_STMT, sql_hdbc, &sql_hstmt);
if(err != SQL_SUCCESS && err != SQL_SUCCESS_WITH_INFO)
{
printf("SQLAllocHandle Error\n");
return 1;
}
//выполнение команды
err = SQLExecDirect(sql_hstmt, (unsigned char*)
"select * from Authors",
SQL_NTS);
if(err != SQL_SUCCESS && err != SQL_SUCCESS_WITH_INFO)
{
printf("SQLExecDirect Error\n");
return 1;
}
SQLCHAR id[30];
SQLINTEGER lid;
SQLCHAR name[30];
SQLINTEGER lname;
SQLCHAR year[30];
SQLINTEGER lyear;
//Связывание указателей со значениями полей текущей записи
err = SQLBindCol(sql_hstmt, 1, SQL_C_CHAR, id, 30, &lid);
err = SQLBindCol(sql_hstmt, 2, SQL_C_CHAR, name, 30, &lname);
err = SQLBindCol(sql_hstmt, 3, SQL_C_CHAR, year, 30, &lyear);
//Вывод заголовка HTML-документа
printf("Content-type: text/html%c%c",10,10);
```

```

printf("<HTML><HEAD><TITLE>MY TEST</TITLE></
HEAD><BODY>");
char * d="%";
//Вывод заголовка таблицы
printf("<Table Width='100%' CellSpacing=10 CellPadding=10
Border=3>",d);
printf("<Caption align=Center>Information from DataBase</Caption>");
printf("<TR><TH> Id </TH><TH Width=100%> Name </TH><TH> Year
</TH></TR>",d);
//Вывод строчек таблицы
while((err = SQLFetch(sql_hstmt)) != SQL_NO_DATA)
{
printf("<TR><TD>%15s </TD><TD>%25s</TD><TD>%10s</TD></TR>",
id, name, year);
}
printf("</Table>");
printf("</BODY></HTML>");
//Удаление служебных переменных
SQLFreeHandle(SQL_HANDLE_STMT, sql_hstmt);
SQLDisconnect(sql_hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, sql_hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, sql_henv);
    return 0;
}

```

В функции `main()` последовательно выполняются следующие операции, установленные интерфейсом ODBC для получения доступа к БД:

- инициализация среды выполнения приложения ODBC;
- подключение к источнику данных;
- выполнение команд SQL для получения требуемых данных;
- освобождение памяти, выделенной под служебные переменные для работы с интерфейсом ODBC.

Для инициализации среды выполнения приложения ODBC используется функция `SQLAllocHandle()`. Следующий код устанавливает переменной `sql_henv` значение идентификатора среды исполнения:

```
err = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &sql_henv);
```

Параметры функции задают следующее:

- тип идентификатора (`SQL_HANDLE_ENV` – идентификатор среды);
- контекст идентификатора;
- адрес переменной, которой присваивается полученный функцией идентификатор.

При успешном завершении функция SQLAllocHandle() возвращает значение **SQL_SUCCESS** или **SQL_SUCCESS_WITH_INFO**; при аварийном завершении – значение **SQL_INVALID_HANDLE** или **SQL_ERROR**.

Для установления параметров среды исполнения используется функция SQLSetEnvAttr():

```
err = SQLSetEnvAttr(sql_henv, SQL_ATTR_ODBC_VERSION,
(SQLPOINTER)SQL_OV_ODBC3, SQL_IS_INTEGER);
```

Здесь параметры функции задают следующее:

- идентификатор среды, для которой выполняется настройка атрибутов;
- тип редактируемого атрибута (значение **SQL_ATTR_ODBC_VERSION** указывает версию драйвера ODBC);
- значение атрибута (для численных значений) или адрес строки атрибута (для атрибутов, заданных в виде текстовой строки);
- тип значения атрибута или длину строки в зависимости от значения третьего параметра (числовое либо строчное соответственно).

Далее в приложении выполняется *инициализация среды соединения*:

```
err = SQLAllocHandle(SQL_HANDLE_DBC, sql_henv, &sql_hdbc);
```

Переменной **sql_hdbc** присваивается идентификатор соединения, имеющий тип **SQL_HANDLE_DBC** и создаваемый в контексте идентификатора среды, хранящегося в переменной **sql_henv**.

Установка соединения с источником данных с использованием имени DSN выполняется с помощью функции SQLConnect():

```
err = SQLConnect(sql_hdbc,
sys_DSN, (SWORD)strlen((const char*)sys_DSN),
user_Name, (SWORD)strlen((const char*)user_Name),
user_Password, (SWORD)strlen((const char*)user_Password));
```

Используемые при вызове функции переменные определены следующим образом:

```
UCHAR sys_DSN[SQL_MAX_DSN_LENGTH + 1] = "Authors";
UCHAR user_Name[MAXNAME] = "";
UCHAR user_Password[MAXNAME] = "";
```

После установления соединения можно задавать запросы по извлечению данных с помощью команд языка SQL. Для выполнения команды на языке SQL нужно получить идентификатор команды, хранящийся в переменной **sql_hstmt**, с помощью функции SQLAllocHandle():

```
err = SQLAllocHandle(SQL_HANDLE_STMT, sql_hdbc, &sql_hstmt);
```

Команда на SQL задается с помощью вызова функции `SQLExecDirect()`, например:

```
err = SQLExecDirect(sql_hstmt, (unsigned char*)
    "select * from Authors", SQL_NTS);
```

Здесь для идентификатора команды, хранящегося в переменной `sql_hstmt`, запускается на выполнение команда SQL `"select * from Authors"`, выбирающая все поля в таблице `Authors`. В этой функции последний аргумент указывает длину строки второго аргумента (SQL-команды), а константа `SQL_NTS` указывает, что длина строки определяется закрывающим ее двоичным нулем.

В случае успешного выполнения команды CGI-модуль получает набор данных из БД. Для извлечения записей из этого набора данных используется функция `SQLBindCol()`, выполняющая связывание указателя на строку символов, хранящуюся в переменной `id`, со значением поля текущей записи:

```
err = SQLBindCol(sql_hstmt, 1, SQL_C_CHAR, id, 30, &lid);
```

Переменные `id` и `lid` определены следующим образом:

```
SQLCHAR id[30];
SQLINTEGER lid;
```

При этом второй параметр функции определяет номер столбца набора данных. Следующие два параметра определяют тип и адрес переменной, которая связывается с полем текущей записи набора данных. Пятый параметр задает максимальный размер связанной переменной. Последний параметр задает адрес переменной, которой присваивается длина данных, извлекаемых функцией `SQLFetch()`:

```
while((err = SQLFetch(sql_hstmt)) != SQL_NO_DATA)
{
    printf("<TR><TD>%15s </TD><TD>%25s</TD><TD>%10s
        </TD></TR>", id, name, year);
}
```

Здесь функция `SQLFetch()` выполняет извлечение очередной записи набора данных. При достижении конца записей она возвращает значение константы `SQL_NO_DATA`. Значение соответствующего поля можно получить путем обращения к связанным переменным внутри цикла.

Для просмотра результатов работы CGI-модуля удобно воспользоваться обозревателем. Для этого нужно создать простейший HTML-документ, содержащий следующую ссылку

```
<A href="http://localhost/scripts/od.exe">
Загрузить CGI-модуль по запросу http://localhost/Scripts/od.exe
</A>
```

Далее нужно откомпилировать CGI-модуль, для чего в оболочке Visual C++6.0 или 5.0 открыть с помощью соответствующего Мастера новое консольное приложение, назвав его "od". Затем перенести в файл "od.cpp" приведенный выше код примера на C++ и откомпилировать его. После успешной компиляции нужно скопировать файл "od.exe" в каталог, где хранятся сценарии (в зависимости от того, под управлением какого сервера предполагается запускать модуль, например для Personal Web Server – "C:\WebShare\Scripts\", для IIS – "C:\InetPub\Scripts\").

Кроме того, следует установить параметры DSN для имени Authors. Для этого в администраторе ODBC нужно создать новое пользовательское имя DSN – "Authors". Причем в настройке параметров этого имени связать этот DSN с демонстрационной БД MS Access, входящей в состав дистрибутива Windows 2000 Server, Authors.mdb и находящейся в каталоге C:\Inetpub\iisamples\SDK\ASP\DATABASE (рис. 16.3).

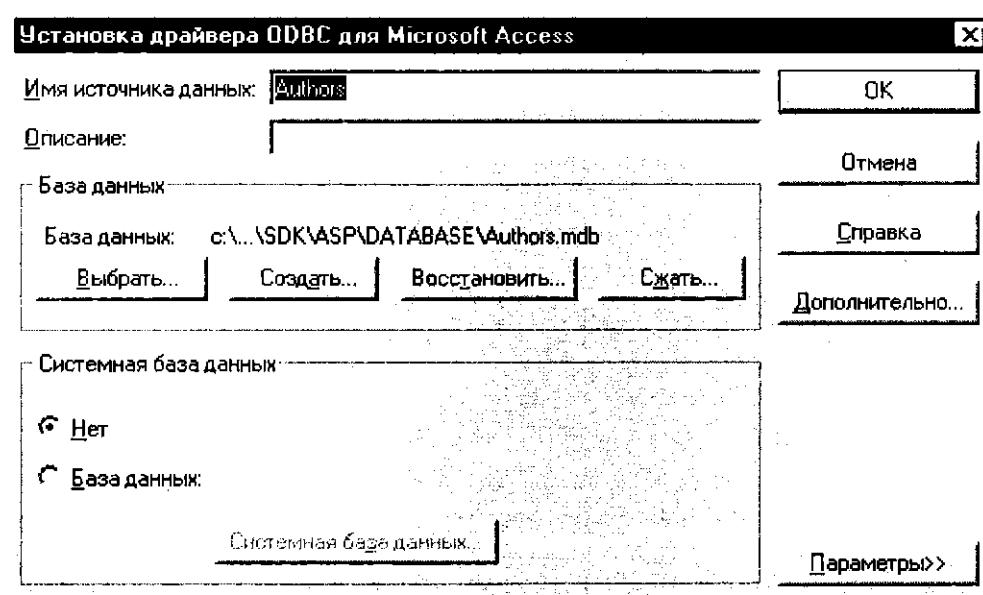


Рис. 16.3. Окно настройки пользовательского DSN администратора ODBC

При необходимости можно создать собственную БД MS Access или другого типа, установив связь с БД с именем DSN описанным выше способом. При этом для соответствия с логикой работы данной программы в БД должно быть не менее трех колонок с такими же именами.

Для получения результата достаточно загрузить приведенный выше HTML-документ в обозреватель и выбрать в нем единственную ссылку. Результат работы модуля "od.exe" приведен на рис. 16.4.

The screenshot shows a Microsoft Internet Explorer window titled "MY TEST - Microsoft Internet Explorer". The address bar displays "http://localhost/scripts/od.exe". The main content area is titled "Information from DataBase". Below the title is a table with four columns: "Price", "Name", and "Year" (with "Id" implied by the context). The table contains four rows of data:

Price	Name	Year
73	Scott Guthrie	1975
114	Shammas, Namir Clement	1954
244	Vaughn, William	1947
611	Bard, Dick	1941

Рис. 16.4. Окно обозревателя с результатом работы модуля “od.exe”

Программа “od.exe” выводит в окно обозревателя содержимое БД “Authors.mdb” в виде таблицы, занимающей всю область окна обозревателя. Причем основное пространство в окне выделено во второй колонке таблицы, что определяется параметром Width=100% заданного для второй ячейки в строке заголовка таблицы.

В частности, в рассматриваемой программе заголовочные теги таблицы выводятся с помощью следующего кода:

```
printf("<Table Width='100%' CellSpacing=10 CellPadding=10 Border=3>",d);
printf("<Caption align=Center>Information from DataBase</Caption>");
printf("<TR><TH> Id </TH><TH Width=100%> Name </TH><TH> Year
</TH></TR>",d);
```

Далее в программе идет код вывода строк таблицы.

Освобождение памяти, выделенной для размещения служебных (системных) переменных, в программе выполняется с помощью функции SQLFreeHandle().

Публикация БД с использованием ADO

ADO является интерфейсом высокого уровня, объединяющим модель объектов для доступа к различным источникам данных на основе использования интерфейса OLE DB. Напомним, что универсальный доступ к различным источникам данных обеспечивается механизмом OLE DB-провайдеров. При этом приложения, использующие интерфейс OLE DB, могут использовать функции провайдера и потребителя данных.

Основные элементы интерфейса OLE DB составляют набор COM-объектов, используемых для установки соединения с базами данных, выполнения команд выборки данных, обработки результатов выполнения этих команд и возникших ошибок.

Рассмотрим основные особенности использования интерфейса ADO для доступа к ODBC-источникам данных (база данных в формате MS Access).

В основе интерфейса ADO лежат принципы, заложенные в объектной модели COM, в которой для получения доступа к данным используются объекты, наделенные свойствами и методами для управления свойствами. В модели ADO реализован набор объектов для установки соединения с источником данных, объекты для выборки необходимых данных (выполнения команд на языке Transact-SQL – версия языка SQL, поддерживаемая фирмой Microsoft), обработки результатов выборки данных и обработки ошибок. Интерфейс ADO предназначен для использования в Интернете для универсального доступа (в том числе удаленного доступа) к различным источникам данных с неизвестной внутренней структурой.

Иерархия объектов интерфейса ADO приведена на рис. 16.5.

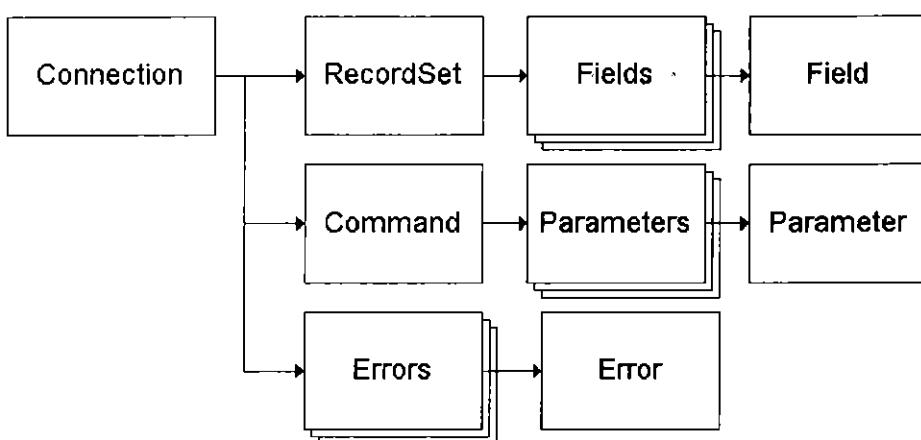


Рис. 16.5. Иерархия объектов интерфейса ADO

Дадим краткую характеристику основным объектам модели ADO. Объект **Connection** используется для установки соединения с источником данных. В рассмотренном ниже примере будет использовано соединение с БД с помощью интерфейса ODBC. В этом случае указывается имя DSN, имя пользователя и пароль. Для объекта **Connection** создаются связанные с ним объекты **Command**, **RecordSet**, **Errors**.

Объект **Command** используется для выполнения команды на языке Transact-SQL для источника данных, доступ к которому открыт с помощью объекта **Connection**. С объектом **Command** могут быть связаны один или несколько объектов **Parameters**, содержащих объекты **Parameter**, используемые для передачи параметров выполнения команд.

Объект **RecordSet** содержит данные, полученные в результате выполнения команд выборки данных. Объект **RecordSet** содержит набор объектов — **Fields**, представляющий собой совокупность объектов **Field**, которые и содержат данные.

В случае возникновения ошибок при выполнении команд создается объект **Error**, который помещается в набор объектов **Errors**.

Пример 1. Использование объектов ADO. Рассмотрим порядок использования объектов ADO на примере следующего модуля CGI, осуществляющего доступ к БД в формате MS Access через провайдера ODBC:

```
//Заголовочные файлы
#include "stdafx.h"
#include <initguid.h>
#include <adoid.h>
#include <adoind.h>
using namespace std;
void main(void)
{
//Инициализация системы COM
CoInitialize(NULL);
//Инициализация служебных строковых переменных
CString csForConnect ("DSN=Authors;UID=;PWD=" );
CString csEmpty ("");
CString csForCommand("select * from Authors");
//Инициализация служебных переменных класса BSTR,
//предназначенного для хранения специальных строк
BSTR bsForConnect= csForConnect.AllocSysString();
BSTR bsEmpty= csEmpty.AllocSysString();
BSTR bsForCommand= csForCommand.AllocSysString();
//Инициализация служебных переменных класса VARIANT
VARIANT vEmpty;
VARIANT vEmptyToo;
vEmpty.vt = VT_ERROR;
```

```
vEmpty.scode = DISP_E_PARAMNOTFOUND;
vEmptyToo.vt = VT_ERROR;
vEmptyToo.scode = DISP_E_PARAMNOTFOUND;
//Инициализация указателей на объекты интерфейса ADO
ADOConnection* con = NULL;
ADOResultset* res = NULL;
ADOCommand* command = NULL;
//Инициализация информационного объекта
HRESULT hres_ok = S_OK;
//Создание объекта Connection
hres_ok = CoCreateInstance(CLSID_CADOConnection, NULL,
    CLSCTX_INPROC_SERVER, IID_IADOConnection, (LPVOID*)&con);
//Запись строки параметров соединения в bsForConnect
if(SUCCEEDED(hres_ok))
    hres_ok = con->put_ConnectionString(bsForConnect);
//Установка соединения с источником данных
if(SUCCEEDED(hres_ok))
    hres_ok = con->Open(bsEmpty, bsEmpty, bsEmpty,
adConnectUnspecified);
//Создание объекта Command
if(SUCCEEDED(hres_ok))
    hres_ok = CoCreateInstance(CLSID_CADOCommand, NULL,
    CLSCTX_INPROC_SERVER, IID_IADOCommand, (LPVOID*)&command);
//Связывание объекта Command с объектом Connection
if(SUCCEEDED(hres_ok))
    hres_ok = command->putref_ActiveConnection(con);
//Запись текста команды в объект Command
if(SUCCEEDED(hres_ok))
    hres_ok = command->put_CommandText(bsForCommand);
//Выполнение команды записи данных
// в переменную res (экземпляр объекта RecordSet)
if(SUCCEEDED(hres_ok))
    hres_ok = command->Execute(&vEmpty, &vEmptyToo, adCmdText, &res);
//Определение переменных для хранения данных
COleVariant s_id;
COleVariant s_name;
COleVariant s_yearBorn;
//Инициализация временных строковых переменных
CString s = "";
CString s1 = "";
//Инициализация специальной логической переменной
VARIANT_BOOL ado_EOF = VARIANT_FALSE;
//Определение конца набора данных
```

```
if(SUCCEEDED(hres_ok))
    hres_ok = res->get_EOF(&ado_EOF);
//Инициализация переменных для хранения указателей на объекты
Fields и Field
ADOFields* adoFields = NULL;
ADOField* id = NULL;
ADOField* name = NULL;
ADOField* yearBorn = NULL;
//Генерация заголовка HTML-документа
s.Format("Content-type: text/html%c%c",10,10);
    cout << (LPCTSTR)s << "\n";
cout << "<HTML><HEAD><TITLE>MY TEST</TITLE></HEAD><BODY>";
//Генерация заголовка таблицы
cout << "<Table Width='100%' CellSpacing=10 CellPadding=10 Border=3>";
cout << "<Caption align=Center>Information from DataBase</Caption>";
cout << "<TR><TH> Id </TH><TH Width=100%> Name of authors</TH>
<TH> YearBorn </TH></TR>";
//Цикл по количеству записей в наборе данных
while(ado_EOF == VARIANT_FALSE)
{
//Извлечение указателя на объект Fields для текущей записи
    hres_ok = res->get_Fields(&adoFields);
    if(!SUCCEEDED(hres_ok)) break;
//Извлечение указателей на объекты Field для объекта Fields
    hres_ok = adoFields->get_Item(COleVariant("Au_ID"), &id);
    if(!SUCCEEDED(hres_ok)) break;
    hres_ok = adoFields->get_Item(COleVariant("author"), &name);
    if(!SUCCEEDED(hres_ok)) break;
    hres_ok = adoFields->get_Item(COleVariant("YearBorn"), &yearBorn);
    if(!SUCCEEDED(hres_ok)) break;
//Извлечение данных
    hres_ok = name->get_Value(&s_name);
    if(!SUCCEEDED(hres_ok)) break;
    hres_ok = yearBorn->get_Value(&s_yearBorn);
    if(!SUCCEEDED(hres_ok)) break;
    hres_ok = id->get_Value(&s_id);
    if(!SUCCEEDED(hres_ok)) break;
//Формирование строки таблицы
    s1=V_BSTR(&s_name);
    s.Format("<TR><TD>%15d</TD><TD>%25s</TD><TD>%10d</TD>",
    V_I4(&s_id),s1,V_I4(&s_yearBorn));
    cout << (LPCTSTR)s << "\n";
//Перемещение указателя текущей записи на следующую запись
```

```
hres_ok = res->MoveNext();
if(!SUCCEEDED(hres_ok))
break;
//Определение конца набора данных
hres_ok = res->get_EOF(&ado_EOF);
}
//Генерация окончания HTML-документа
cout << "</Table>";
cout << "</BODY></HTML>";
//Освобождение ресурсов
res->Close();
con->Close();
CoUninitialize();
SysFreeString(bsForConnect);
SysFreeString(bsEmpty);
SysFreeString(bsForCommand);
return ;
}
```

Этот модуль посылает в стандартный поток вывода HTML-документ, аналогичный документу, генерируемому модулем, использующим интерфейс ODBC (см. раздел 16.1.4).

Для запуска CGI-модуля, использующего интерфейс ADO, нужно выполнить такую же последовательность действий, как и указанную в разделе 16.1.4.

Поясним операции, выполняемые приведенным модулем. В начале модуля подключаются заголовочные файлы. Файл "stdafx.h" генерируется средой Visual C++ и содержит подключение группы заголовочных файлов, входящих в состав стандартных библиотек, которые определяют тип проекта.

В первых строках функции main() производится инициализация служебных переменных. Для инициализации системы COMобъектов используется функция CoInitialize(NULL).

Затем производится инициализация служебных переменных и создание объекта Connection:

```
ADOConnection* con = NULL;
HRESULT hres_ok = S_OK;
hres_ok = CoCreateInstance(CLSID_CADOConnection, NULL,
CLSTX_INPROC_SERVER, IID_IADOConnection, (LPVOID*)&con);
```

Так как устанавливается соединение с ODBC-источником данных, то необходимо задать параметры DSN-соединения. Для этого используется строковая переменная csForConnect:

```
CString csForConnect ("DSN=Authors;UID=;PWD=;" );
BSTR bsForConnect= csForConnect.AllocSysString();
```

При этом DSN=Authors, а имя пользователя и пароль задаются пустыми с помощью параметров соединения — UID и PWD. Специальные информационные строки типа BSTR используются методами объектов интерфейса ADO. Для инициализации таких строковых переменных используется функция AllocSysString(), являющаяся членом класса Cstring.

Для установки соединения с источником данных нужно записать строку параметров соединения, хранящуюся в переменной bsForConnect, в объект Connection. Для этого используется метод put_ConnectionString() объекта Connection:

```
hres_ok = con->put_ConnectionString(bsForConnect);
```

Непосредственно соединение с источником данных осуществляется с помощью метода Open():

```
hres_ok = con->Open(bsEmpty, bsEmpty, bsEmpty,
                     adConnectUnspecified);
```

Здесь переменная bsEmpty является пустой строкой, так как все параметры соединения заданы в строке bsForConnect, а константа adConnectUnspecified определяет синхронный режим открытия соединения с источником данных.

Для выборки данных нужно инициализировать объект Command, связать его с объектом Connection методом putref_ActiveConnection(), записать текст команды с помощью метода put_CommandText() и выполнить метод Execute() объекта Command:

```
ADOCommand* command = NULL;
CoCreateInstance(CLSID_CADOCommand, NULL,
                 CLSCTX_INPROC_SERVER, IID_IADOCCommand,
                 (LPVOID*)&command);
command->putref_ActiveConnection(con);
command->put_CommandText(bsForCommand);
command->Execute(&vEmpty, &vEmptyToo, adCmdText, &res);
```

Здесь значением переменной bsForCommand является код на Transact-SQL, позволяющий выбирать все поля в таблице Authors — “select * from Authors”. Переменные vEmpty и vEmptyToo типа VARIANT имеют следующие значения полей:

```
vEmpty.vt = VT_ERROR;
vEmpty.scode = DISP_E_PARAMNOTFOUND;
vEmptyToo.vt = VT_ERROR;
vEmptyToo.scode = DISP_E_PARAMNOTFOUND;
```

Параметр adCmdText устанавливает, что аргумент метода put_CommandText() — переменная bsForCommand является строкой на языке Transact-SQL. Результирующий набор данных записывается по адресу, заданному переменной res, определенной как

```
ADOResultset* res = NULL;
```

Для извлечения данных из объекта **Recordset** организован цикл, в котором последовательно просматривается каждая строка результирующего набора данных. В теле цикла анализируются поля из набора данных, соответствующие текущей строке. Для этого используется указатель на текущую строку набора данных. С помощью строки

```
res->MoveNext();
```

выполняется переход к следующей записи набора данных.

Для выхода из цикла используется переменная **ado_EOF**, которой присваивают значение логической переменной с помощью метода **get_EOF(&ado_EOF)**, принадлежащего объекту **Recordset**. Выход из цикла осуществляется при достижении конца набора данных, то есть при невыполнении следующего условия

```
ado_EOF == VARIANT_FALSE
```

Здесь **VARIANT_FALSE** – константа, значение которой функция **get_EOF()** возвращает при успешном перемещении указателя на следующую запись в наборе данных.

Для извлечения данных текущей записи выполняются следующие действия:

- получение указателя на объект **Fields** для текущей записи;
- получение указателей на объекты **Field** для объекта **Fields**;
- получение данных из объекта **Field**.

Для получения указателя на объект **Fields** для текущей записи используется метод объекта **Recordset**

```
get_Fields(&adoFields)
```

где **adoFields** – переменная типа **ADOFIELDS**.

Для получения указателей на объекты **Field** используется следующий код:

```
hres_ok = adoFields->get_Item(COleVariant("Au_ID"), &id);
hres_ok = adoFields->get_Item(COleVariant("author"), &name);
hres_ok = adoFields->get_Item(COleVariant("YearBorn"), &yearBorn);
```

Здесь переменные **id**, **name**, **yearBorn** получают указатели на объекты **Field**, в которых содержатся данные для соответствующего столбца и текущей записи из набора данных и определены следующим образом:

```
ADOField* id = NULL;
ADOField* name = NULL;
ADOField* yearBorn = NULL;
```

Для получения данных из объекта **Field** используется метод **get_Value()** объекта **Fields**:

```
hres_ok = name->get_Value(&s_name);
hres_ok = yearBorn->get_Value(&s_yearBorn);
hres_ok = id->get_Value(&s_id);
```

Формирование строки таблицы выполняет следующий код:

```
s1=V_BSTR(&s_name);
s.Format("<TR><TD>%15d</TD><TD>%25s</TD><TD>%10d</TD>",
V_I4(&s_id),s1,V_I4(&s_yearBorn));
cout << (LPCTSTR)s << "\n";
```

Здесь метод `Format()` используется для преобразования целочисленных переменных в строковые переменные, а системные функции `V_BSTR()` и `V_I4()` используются для приведения типов.

16.2. Интерфейс программирования серверных приложений ISAPI

Общая характеристика интерфейса

Перспективным направлением развития технологий создания моделей расширения сервера является интерфейс (протокол) программирования серверных приложений ISAPI. Этот интерфейс разработан для преодоления ограничений и недостатков интерфейса CGI. В отличие от общего интерфейса CGI интерфейс ISAPI предусматривает реализацию модулей расширения сервера в виде *динамической библиотеки*. При этом для обработки каждого запроса обозревателя к модулю расширения сервера ISAPI не создается отдельный процесс, а для обработки запроса вызываются функции модуля ISAPI и один раз загружаются в память при первом обращении.

Существует несколько вариантов реализации интерфейсов серверных расширений. Наибольшее распространение получили два из них – NSAPI компании Netscape и ISAPI фирмы Microsoft. Из-за ограниченности объема мы рассмотрим возможности интерфейса ISAPI. Приложения, построенные с использованием интерфейса ISAPI, имеют в основе аналогичные принципы взаимодействия с сервером, что и CGI-модули. Отличия этих интерфейсов заключаются в реализации обмена данными между модулем и сервером.

Модули расширения ISAPI, как и модули CGI, получают от сервера данные, которые передает модулю обозреватель, обрабатывают эти данные и динамически формируют HTML-документ. Однако вместо чтения содержимого переменных окружения и стандартного потока ввода модуль расширения ISAPI получает данные при помощи предназначенных для этого функций. Аналогично, вместо записи выходных данных в стандартный поток вывода модуль расширения ISAPI вызывает специальные функции.

Модули ISAPI в зависимости от назначения подразделяются на две группы: обычные модули расширения ISAPI и фильтры ISAPI. Первая группа аналогично, как и CGI-модули расширения сервера, предназначена для динамического формирования HTML-документов. Фильтры ISAPI предназначены для

решения нестандартных задач по контролю всех данных, проходящих через сервер. Например, кодирование данных, сбор статистической информации об использовании ресурсов сервера и т. д. Разработка ISAPI-фильтров нами не рассматривается, так как эта тема требует написания отдельной книги.

Рассмотрим основные принципы создания и работы модулей расширений ISAPI. Модуль ISAPI реализуется в виде библиотеки DLL. Эта библиотека загружается в одно адресное пространство с Web-сервером при первом обращении к ней из обозревателя. Причем обращение к ISAPI-модулю организуется аналогично используемым методам вызова CGI-модулей расширения—из форм или ссылок. ISAPI-модуль выполняется в рамках единого адресного пространства сервера с программными ресурсами сервера. Такая организация ISAPI-модуля позволяет значительно повысить производительность сервера в целом по сравнению с сервером, использующим CGI-модули.

Преимущество ISAPI-интерфейса особенно заметно при одновременном активном использовании модуля расширения несколькими пользователями.

Отметим, что CGI-модули являются более надежным средством с точки зрения устойчивости Web-сервера к сбоям. Так, при возникновении ошибки в ISAPI-модуле может произойти аварийное завершение всего Web-сервера, тогда как в случае возникновения ошибки в CGI-модуле будет завершен только процесс. По этим причинам особенности отладки ISAPI-модуля состоят в том, что для перезапуска ISAPI-модуля требуется его переименование или перезапуск Web-сервера.

Рассмотрим структуру модулей ISAPI. Библиотека DLL, представляющая собой ISAPI-модуль, экспортирует как минимум две функции с именами `GetExtensionVersion()` и `HttpExtensionProc()`. Кроме этих функций, ISAPI-модуль может использовать функцию `TerminateExtension()`, которой передается управление перед выгрузкой ISAPI-модуля из памяти. Эта функция освобождает ресурсы, используемые ISAPI-модулем.

Функция `GetExtensionVersion()` предназначена для того, чтобы ISAPI-модуль мог сообщить серверу версию спецификации ISAPI-интерфейса, структуру описания ISAPI-модуля и имеет следующий прототип:

```
extern "C" BOOL WINAPI GetExtensionVersion(HSE_VERSION_INFO *pVer);
```

где переменная `pVer` является указателем на структуру типа `HSE_VERSION_INFO` и определена в файле `httpext.h` следующим образом:

```
#define HSE_MAX_EXT_DLL_NAME_LEN256
typedef struct _HSE_VERSION_INFO
{
    DWORD dwExtensionVersion; // версия спецификации интерфейса ISAPI
    CHAR lpszExtensionDesc[HSE_HAX_EXT_DLL_N AME_LEN]; // строка
    //описания модуля ISAPI
} HSE_VERSION_INFO, *LPHSE_VERSION_INFO;
```

Функция **HttpExtensionProc()** выполняет основную работу ISAPI-модуля и используется для обмена данными между ISAPI-модулем и сервером. Функция **HttpExtensionProc()** имеет следующий прототип:

```
extern "C" DWORD WINAPI HttpExtensionProc(
    EXTENSION_CONTROL_BLOCK *pECB);
```

где переменная **pECB** является указателем на управляющую структуру (блок) **EXTENSION_CONTROL_BLOCK**, определенную в файле **httpext.h** следующим образом:

```
typedef struct EXTENSION_CONTROL_BLOCK {
    DWORD cbSize; // размер блока в байтах
    DWORD dwVersion; // версия спецификации интерфейса ISAPI
    HCONN ConnID; // идентификатор соединения
    DWORD dwHttpStatusCode; // код состояния HTTP
    CHAR LpszLogData[HSE_LOG_BUFFER_LEN]; // текстовая строка,
    // содержащая данные протоколирования
    LPSTR LpszMethod; // переменная окружения REQUEST_METHOD
    LPSTR LpszQueryString; // переменная окружения QUERY_STRING
    LPSTR LpszPathInfo; // переменная окружения PATH_INFO
    LPSTR LpszPathTranslated; // переменная окружения PATH_TRANSLATED
    DWORD cbTotalBytes; // полный размер данных, полученных от обозревателя
    DWORD cbAvailable; // размер доступного блока данных
    LPBYTE LpbData; // указатель на доступный блок данных
    // размером cbAvailable байт
    LPSTR LpszContentType; // тип принятых данных
    //Функция GetServerVariable для получения значения переменных окружения
    BOOL (WINAPI * GetServerVarlable)(HCONN hConn,LPSTR
        LpszVarlableName, LPVOID LpvBuffer, LPDWORD LpdwSize);
    // Функция WriteClient для посылки данных обзревателю
    BOOL (WINAPI * WriteClient)(HCONN ConnID,LPVOID Buffer,
        LPDWORD LpdwBytes, DWORD dwReserved);
    // Функция ReadClient для получения данных от посетителя
    BOOL (WINAPI * ReadClient) (HCONN ConnID, LPVOID LpvBuffer,
        LPDWORD LpdwSize);
    // Служебная функция
    ServerSupportFunction BOOL (WINAPI * ServerSupportFunction)(HCONN
        hConn,DWORD dwHSERRequest, LPVOID LpvBuffer,LPDWORD
        LpdwSize, LPDWORD LpdwDataType);
} EXTENSION_CONTROL_BLOCK, *LPEXTENSION_CONTROL_BLOCK;
```

Поясним особенности использования основных полей этой структуры данных.

Поле **cbTotalBytes** предназначено для записи суммарного количества байт данных, получаемых от обозревателя. Блок данных размером не более 48 кб считывается сервером автоматически. Эти данные становятся доступны при вызове функции **HttpExtensionProc()**. Но полностью данные дочитываются в цикле при помощи функции **ReadClient()**.

Поле **cbAvailable** предназначено для записи размера блока данных, полученных автоматически от обозревателя (размером не более 48 кб).

Поле **LpbData** содержит указатель на область памяти, в которую сервером записан полученный от обозревателя блок данных размером **cbAvailable** байтов.

В управляющем блоке, кроме данных, находятся следующие указатели на методы:

Поле **GetServerVariable** содержит указатель на функцию, средствами которой ISAPI-модуль определяет значения переменных среды.

Поле **WriteClient** содержит указатель на функцию, используемую для отправки данных обозревателю, в отличие от интерфейса CGI, при котором модуль расширения помещает результат в стандартный поток вывода.

Поле **ReadClient** содержит указатель на функцию, предназначенную для считывания данных в буфер предварительного чтения, имеющий адрес **LpbData** и размер, не превышающий 48 кб.

Поле **ServerSupportFunction** содержит указатель на функцию, предназначенную для пересылки стандартного заголовка протокола HTTP и других действий.

Структуры данных для разработки ISAPI-модуля

В системе программирования VisualC++6.0 для создания ISAPI-модуля разработаны следующие классы:

- **CHtmlStream** – класс потока, используемый для управления выходным потоком, в который помещается HTML-документ;
- **CHttpServerContext** – вспомогательный класс контекста для обмена данными модуля с сервером;
- **CHttpServer** – базовый класс для создания модуля ISAPI.

Эти классы определены в файле "asfisapi.h".

Основные методы и свойства класса **CHttpServerContext** определены следующим образом:

```
class CHttpServerContext
{
public:
    CHttpServerContext(EXTENSION_CONTROL_BLOCK* pECB); //Конструктор
    virtual ~CHttpServerContext(); //Деструктор
    // Функции, используемые для обмена данными с сервером
    BOOL GetServerVariable(LPTSTR lpszVariableName, LPVOID lpvBuffer,
        LPDWORD lpdwSize); //Функция для получения переменных окружения
```

```
BOOL WriteClient(LPVOID IpvBuffer, LPDWORD IpdwBytes, DWORD
    dwReserved = 0); //Функция для записи ответа обозревателю
BOOL ReadClient(LPVOID IpvBuffer, LPDWORD IpdwSize); //Функция
// для чтения из потока данных, передаваемых обозревателем серверу
BOOL ServerSupportFunction(DWORD dwHSERRequest, LPVOID
    IpvBuffer, LPDWORD IpdwSize, LPDWORD IpdwDataType); //Функция
//для пересылки стандартного заголовка протокола HTTP и других действий
...
CHttpServerContext& operator<<(LPCTSTR psz); //Переопределение
//оператора "<<", используемого для записи строки в выходной поток данных
...
// Основные свойства
public:
...
    DWORD m_dwStatusCode; //Код возврата
    EXTENSION_CONTROL_BLOCK* const m_pECB; //Указатель на блок
//управления модулем
    CHtmlStream* m_pStream; //Указатель на поток
...
};

class CHttpServer
{
public:
    CHttpServer(TCHAR cDelimiter = '&'); //Конструктор
    virtual ~CHttpServer(); //Деструктор
...
// Функции для вывода фрагментов HTML-документа в выходной поток
    virtual void EndContent(CHttpServerContext* pCtxt) const; //Функция,
//заканчивающая вывод HTML-документа
    virtual void StartContent(CHttpServerContext* pCtxt) const; //Функция,
//используемая для вывода заголовка о типе данных и других действий
    virtual void WriteTitle(CHttpServerContext* pCtxt) const; //Функция
//для вывода заголовка HTML-документа
    virtual LPCTSTR GetTitle() const; //Функция,
//возвращающая заголовок HTML-документа
    void AddHeader(CHttpServerContext* pCtxt, LPCTSTR pszString) const;
//Функция, выводящая заголовочную часть HTML-документа
    virtual BOOL TerminateExtension(DWORD dwFlags); //Функция,
//освобождающая ресурсы, используемые ISAPI-модулем
    virtual DWORD HttpExtensionProc(EXTENSION_CONTROL_BLOCK *pECB);
// Главная функция, в которой формируется HTML-документ
    virtual BOOL GetExtensionVersion(HSE_VERSION_INFO *pVer);
```

```
//Функция, записывающая версию спецификации ISAPI-интерфейса
//и строку описания ISAPI-модуля
...
};
```

На основе этих классов Мастера, используемые в среде VisualC++6.0, формируют разрабатываемый ISAPI-модуль, в котором создается собственный класс, наследующий класс **CHttpServer**. Объект этого класса объединяет все свойства и методы, используемые для обмена информацией между сервером и модулем. Кроме того, с помощью функций этого объекта подменяются основные функции, которые должен экспортить ISAPI-модуль — **GetExtensionVersion()** и **HttpExtensionProc()**.

Получение данных ISAPI-модулем

ISAPI-модуль имеет более сложную организацию чем модуль CGI. Для получения переданных от обозревателя данных ISAPI-модуль использует свойства и методы специальных объектов типа **ServerContext** и **EXTENSION_CONTROL_BLOCK**, имеющих аналогичные поля и методы для извлечения данных. В полях этих объектов находятся значения переменных окружения. Кроме того, с помощью методов этих объектов можно получить данные непосредственно от обозревателя. Например, для получения значений переменных окружения применяется следующий код, использующий функцию **GetServerVariable()**:

```
char par[10];
dwSize = 10;
pCtxt->m_pECB->GetServerVariable(pCtxt->m_pECB->ConnID,
(LPSTR)"PATH_INFO", (LPVOID)par, &dwSize);
```

где **pCtxt** — указатель на контекст сервера типа **CHttpServerContext**,
m_pECB — указатель на блок типа **EXTENSION_CONTROL_BLOCK**.

Через параметр **ConnID** передается идентификатор соединения, извлекаемый из объекта **EXTENSION_CONTROL_BLOCK**.

Второй параметр содержит указатель на строку имени переменной окружения.

Третий параметр содержит указатель на буфер — **par**, в который помещается значение соответствующей переменной окружения а размер этой переменной помещается в параметр **dwSize**.

Приведем значения имен основных переменных окружения, которые можно получить с помощью этой функции:

- **AUTH_TYPE** — содержит тип идентификации, используемый сервером;
- **HTTP_ACCEPT** — типы данных MIME, поддерживаемые обозревателем;
- **CONTENT_LENGTH** — количество байтов данных, передаваемых обозревателем;
- **CONTENT_TYPE** — тип данных, передаваемых обозревателем;

- **PATH_INFO** – данные, находящиеся в URL после имени модуля (разделенные символом “/”) либо путь к виртуальному каталогу, в котором находится ISAPI-модуль;
- **PATH_TRANSLATED** – физический путь ISAPI-модулю;
- **QUERY_STRING** – данные, передаваемые обозревателем, и находящиеся после адреса URL ISAPI-модуля после символа “?”;
- **REMOTE_ADDR** – адрес IP-узла обозревателя;
- **REHOTE_HOST** – доменное имя узла обозревателя или адрес IP;
- **REMOTE_USER** – имя пользователя, используемое обозревателем для аутентификации;
- **REQUEST_HETHOD** – используемый метод передачи данных от обозревателя к серверу;
- **SCRIPT_NAME** – путь к виртуальному каталогу и имя ISAPI-модуля;
- **SERVER_NAME** – доменное имя Web-сервера или адрес IP Web-сервера, если доменное имя недоступно или не определено;
- **SERVER_PROTOCOL** – название и версия протокола выполнения запроса к ISAPI-модулю;
- **SERVER_PORT** – номер порта, с которого обозреватель посыпает запросы Web-серверу;
- **SERVER_SOFTWARE** – название и версия программного обеспечения Web-сервера;
- **ALL_HTTP** – значения всех переменных, относящихся к протоколу HTTP и отделенных друг от друга символом двоеточия “:”, а сами переменные разделены символом перевода строки. Сюда относят переменные, **HTTP_ACCEPT**, **HTTP_CONNECTION**, **HTTP_SER_AGENT** и т. д.

Для отделения значений переменных могут использоваться функции, приведенные в случае использования CGI-модулей в разделе 16.1.

Заметим, что названия этих строк почти совпадают с названиями переменных среды, создаваемых для программ CGI, однако совпадение все же не полное.

Еще одна функция **ReadClient()** используется для чтения данных, передаваемых обозревателем через поток:

```
BOOL (WINAPI * ReadClient) (HCONN ConnID, LPVOID LpvBuffer,  
LPDWORD LpdwSize);
```

где **ConnID** – идентификатор канала, который можно получить через структуру **EXTENSION_CONTROL_BLOCK**, **LpvBuffer** – адрес буфера, в который помещаются данные.

Отправка данных ISAPI-модулем

ISAPI-модуль записывает выходные данные методов не через стандартный поток вывода, как в случае с CGI-модулем, а с использованием методов объектов типа **ServerContext** и **EXTENSION_CONTROL_BLOCK**: **WriteClient()**,

ServerSupportFunction() либо с помощью переопределенной операции “<<” вывода в поток для разных типов данных.

Прототип функции WriteClient() имеет следующий вид:

```
BOOL (WINAPI * WriteClient)(HCONN ConnID,LPVOID Buffer, LPDWORD LpdwBytes, DWORD dwReserved);
```

Функция WriteClient() выводит данные из буфера Buffer, размер которого должен быть записан в переменную LpdwBytes. Параметр dwReserved зарезервирован для расширений возможностей функции. В параметре ConnID находится идентификатор соединения.

Функция ServerSupportFunction() имеет следующий прототип:

```
ServerSupportFunction BOOL (WINAPI *  
ServerSupportFunction)(HCONN hConn,DWORD dwHSERRequest,  
LPVOID LpvBuffer,LPDWORD LpdwSize, LPDWORD LpdwDataType);
```

где hConn – идентификатор соединения (канала); dwHSERRequest – код запроса, задающий тип выполняемой функцией операции; LpvBuffer – адрес буфера; LpdwSize – размер буфера; LpdwDataType – дополнительные данные, добавляемые к заголовку HTTP-документа.

Поясним основные способы использования этой функции, определяемые значением параметра dwHSERRequest.

- HSE_REQ_SEND_RESPONSE_HEADER – пересылка обозревателю заголовка HTTP, определяемого параметром LpdwDataType. Отметим, что функция ServerSupportFunction() пересыпает текстовые данные, а функция WriteClient() может использоваться и для пересылки двоичных данных.
- HSE_REQ_SEND_URL – пересылка обозревателю данных, находящихся по другому адресу URL, задаваемому в виде текстовой строки, через параметр LpvBuffer. Параметр LpdwDataType в этом случае не используется.
- HSE_REQ_SEND_URL_REDIRECT_RESP – отправка сообщения с номером 302 (URL Redirect) по адресу URL, задаваемому в виде текстовой строки, через параметр LpvBuffer.
- HSE_REQ_MAP_URL_TO_PATH – преобразование логического адреса URL в физический. При этом логический адрес передается через параметр LpvBuffer, в который и записывается физический адрес. В переменную LpdwSize заносится размер строки результата преобразования.
- HSE_REQ_DONE_WITH_SESSION – указание серверу о завершении обработки в случае, если ISAPI-модуль оставляет соединение открытым для выполнения длительной обработки данных. При этом все параметры функции игнорируются.

Пример публикации БД с использованием ADO

Рассмотрим особенности использования протокола ISAPI на примере создания ISAPI-модуля, который публикует информацию из БД с использованием интерфейса ADO на HTML-странице.

Для создания собственного ISAPI-модуля в среде Visual C++6.0 нужно выполнить следующие действия:

- запустить среду Visual C++6.0 (предварительно она должна быть установлена);
- выбрать в главном меню интерактивной среды пункт File/New;
- в появившемся окне (рис. 16.6) выбрать тип создаваемого приложения. В нашем случае – ISAPI Extension Wizard, набрать имя проекта – my и выбрать каталог, где будут размещаться файлы проекта;
- следовать указаниям Мастера ISAPI расширения сервера (появятся еще два окна, в которых достаточно для первого раза нажать кнопки Finish и OK соответственно в первом и втором окне).

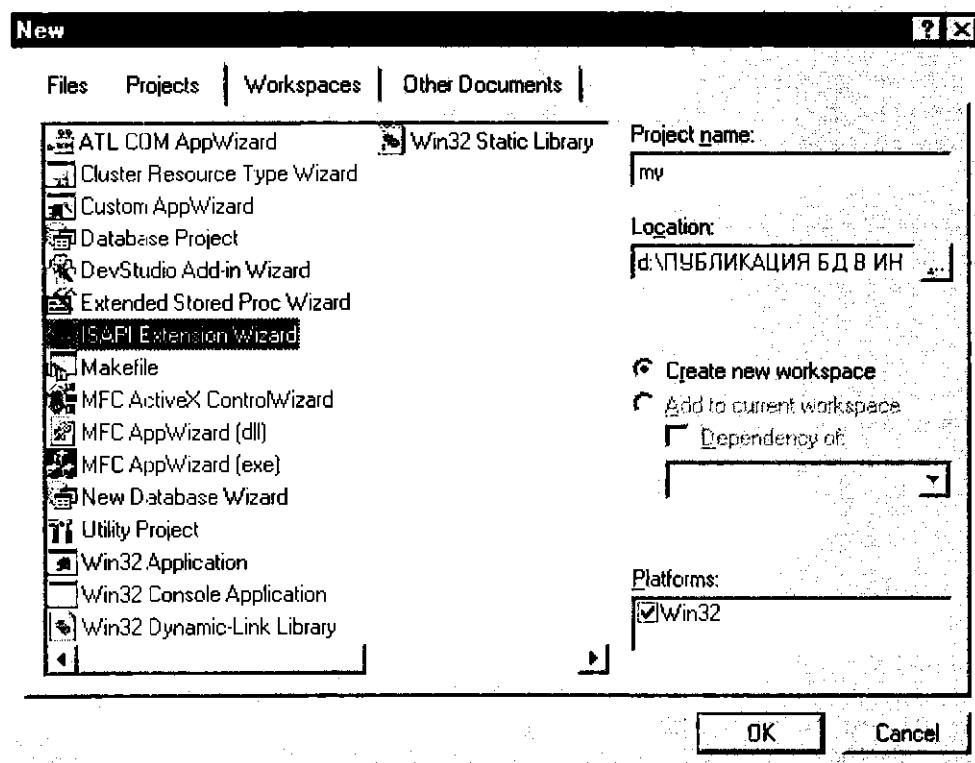


Рис. 16.6. Окно выбора типа создаваемого приложения

После выполнения этих действий будет создан проект, основным рабочим файлом которого будет выступать "my.cpp". С этим файлом связан заголовочный файл "my.h", в котором находится определение класса **CMyExtension**, являющегося потомком класса **CHttpServer**, предназначенного для создания на его основе ISAPI-модулей расширения сервера. В заголовочном файле "my.h" будет создано следующее определение класса:

```
class CMyExtension : public CHttpServer
{
public:
    CMyExtension();
    ~CMyExtension();

// Overrides
// ClassWizard generated virtual function overrides
// NOTE — the ClassWizard will add and remove member functions here.
// DO NOT EDIT what you see in these blocks of generated code !
//{{AFX_VIRTUAL(CMyExtension)
public:
    virtual BOOL GetExtensionVersion(HSE_VERSION_INFO* pVer);
//}}AFX_VIRTUAL
    virtual BOOL TerminateExtension(DWORD dwFlags);
// TODO: Add handlers for your commands here. For example:
    void Default(CHttpServerContext* pCtx);
    DECLARE_PARSE_MAP()
//{{AFX_MSG(CMyExtension)
//}}AFX_MSG
};

};
```

Для использования протокола ADO в файл “ту.cpp”, как и в предыдущем разделе, нужно поместить в исходный текст следующие директивы подключения заголовочных файлов:

```
#include "stdafx.h"  
#include <initguid.h>  
#include "adoid.h"  
#include "adoint.h"
```

Основные особенности использования протокола ADO рассмотрены в разделе 16.1.

Для создания простейшего ISAPI-модуля следует поместить код, генерирующий HTML-документ, в функцию Default(CHttpServerContext* pCtx).

```
void CMyExtension::Default(CHttpServerContext* pCtx)
{
    StartContent(pCtx);
```

```
    WriteTitle(pCtxt);
    //Строковая переменная, используемая для хранения
    //формируемого HTML-документа
    CHAR resBuff[4096];
    CHAR tmpBuff[4096];
    DWORD dwSize;
    // Запись в буфер заголовка HTTP
    wsprintf(resBuff,
        "<HTML><HEAD><TITLE> ISAPI-модуль </TITLE></HEAD>\n"
        "<BODY BGCOLOR=#FFFFFF><H2>Работает ISAPI-модуль!</H2>\n");
    // Добавление разделительной линии
    strcat(resBuff, "<HR>");
    // Вывод названия метода передачи данных
    wsprintf(tmpBuff, "<BR>Используется метод: %s", pCtxt->m_pECB-
        >lpSzMethod);
    strcat(resBuff, tmpBuff);
    // Отображение содержимого переменной PATH_INFO
    strcat(resBuff, "<BR>Значение переменной PATH_INFO:");
    char par[10];
    dwSize = 10;
    pCtxt->m_pECB->GetServerVariable(pCtxt->m_pECB->ConnID,
        (LPSTR)"PATH_INFO", (LPVOID)par, &dwSize);
    strcat(resBuff, par);
    // Вывод окончания документа HTML
    //strcat(resBuff, "</BODY></HTML>");
    *pCtxt << _T(resBuff);
    //Инициализация системы COM
    CoInitialize(NULL);
    //Инициализация служебных строковых переменных
    CString csForConnect ("DSN=Authors;UID=;PWD=" );
    CString csEmpty ("");
    CString csForCommand("select * from Authors where author='Simpson, Alan'");
    CString csForCommand1("select * from Authors where author='Jones, Edward'");
    //Инициализация служебных переменных класса BSTR,
    //предназначенного для хранения специальных строк
    BSTR bsForConnect= csForConnect.AllocSysString();
    BSTR bsEmpty= csEmpty.AllocSysString();
    BSTR bsForCommand;
    if (!strcmp(par,"/name1"))
        bsForCommand= csForCommand.AllocSysString();
    else
```

```
bsForCommand= csForCommand1.AllocSysString();
//Инициализация служебных переменных класса VARIANT
VARIANT vEmpty;
VARIANT vEmptyToo;
vEmpty.vt = VT_ERROR;
vEmpty.scode = DISP_E_PARAMNOTFOUND;
vEmptyToo.vt = VT_ERROR;
vEmptyToo.scode = DISP_E_PARAMNOTFOUND;
//Инициализация указателей на объекты интерфейса ADO
ADOConnection* con = NULL;
ADOResultset* res = NULL;
ADOCommand* command = NULL;
//Инициализация информационного объекта
HRESULT hres_ok = S_OK;
//Создание объекта Connection
hres_ok = CoCreateInstance(CLSID_CADOConnection, NULL,
CLSID_INPROC_SERVER, IID_IADOConnection, (LPVOID*)&con);
//Запись строки параметров соединения в bsForConnect
if(SUCCEEDED(hres_ok))
hres_ok = con->put_ConnectionString(bsForConnect);
//Установка соединения с источником данных
if(SUCCEEDED(hres_ok))
hres_ok = con->Open(bsEmpty, bsEmpty, bsEmpty,
adConnectUnspecified);
//Создание объекта Command
if(SUCCEEDED(hres_ok))
hres_ok = CoCreateInstance(CLSID_CADOCommand, NULL,
CLSID_INPROC_SERVER, IID_IADOCommand,
(LPVOID*)&command);
//Связывание объекта Command с объектом Connection
if(SUCCEEDED(hres_ok))
hres_ok = command->putref_ActiveConnection(con);
//Запись текста команды в объект Command
if(SUCCEEDED(hres_ok))
hres_ok = command->put_CommandText(bsForCommand);
//Выполнение команды для записи данных в
//переменную res (экземпляр объекта RecordSet)
if(SUCCEEDED(hres_ok))
hres_ok = command->Execute(&vEmpty, &vEmptyToo, adCmdText, &res);
//Определение переменных для хранения данных
COleVariant s_name;
```

```
COleVariant s_yearBorn;
//Инициализация временных строковых переменных
CString s = "";
CString s1 = "";
//Инициализация специальной логической переменной
VARIANT_BOOL ado_EOF = VARIANT_FALSE;
//Определение конца набора данных
if(SUCCEEDED(hres_ok))
    hres_ok = res->get_EOF(&ado_EOF);
//Инициализация переменных для хранения указателей на объекты
Fields и Field
ADOFIELDS* adoFields = NULL;
ADOFIELD* name = NULL;
ADOFIELD* yearBorn = NULL;
//Генерация заголовка таблицы
*pCtxt << _T("<Table Width='100%' CellSpacing=10 CellPadding=10
    Border=3>");
*pCtxt << _T("<Caption align=Center>Информация из базы данных
    </Caption>");
*pCtxt << _T("<TR><TH Width=100%> Name of authors</TH>
    <TH> YearBorn </TH></TR>");
//Цикл по количеству записей в наборе данных
while(ado_EOF == VARIANT_FALSE)
{
//Извлечение указателя на объект Fields для текущей записи
    hres_ok = res->get_Fields(&adoFields);
    if(!SUCCEEDED(hres_ok)) break;
//Извлечение указателей на объекты Field для объекта Fields
    hres_ok = adoFields->get_Item(COleVariant("author"), &name);
    if(!SUCCEEDED(hres_ok)) break;
    hres_ok = adoFields->get_Item(COleVariant("YearBorn"), &yearBorn);
    if(!SUCCEEDED(hres_ok)) break;
//Извлечение данных
    hres_ok = name->get_Value(&s_name);
    if(!SUCCEEDED(hres_ok)) break;
    hres_ok = yearBorn->get_Value(&s_yearBorn);
    if(!SUCCEEDED(hres_ok)) break;
//Формирование строки таблицы
    s1=V_BSTR(&s_name);
    s.Format("<TR><TD>%25s</TD><TD>%10d</TD>",
    s1,V_I4(&s_yearBorn));
```

```
*pCtxt << _T((LPCTSTR)s );
//Перемещение указателя текущей записи на следующую запись
hres_ok = res->MoveNext();
if(!SUCCEEDED(hres_ok))
break;
//Определение конца набора данных
hres_ok = res->get_EOF(&ado_EOF);
}
//Генерация окончания HTML-документа
*pCtxt << _T("</Table>");
*pCtxt << _T("</BODY></HTML>");
//Освобождение ресурсов
res->Close();
con->Close();
CoUninitialize();
SysFreeString(bsForConnect);
SysFreeString(bsEmpty);
SysFreeString(bsForCommand);
EndContent(pCtxt);
}
```

Текст функции GetExtensionVersion:

```
BOOL CMyExtension::GetExtensionVersion(HSE_VERSION_INFO* pVer)
{
// Инициализация структуры типа HSE_VERSION_INFO
    CHtmlServer::GetExtensionVersion(pVer);
    TCHAR sz[HSE_MAX_EXT_DLL_NAME_LEN+1]; // Определение
//строки описания
    ISAPIVERIFY(::LoadString(AfxGetResourceHandle(),
        IDS_SERVER, sz, HSE_MAX_EXT_DLL_NAME_LEN));
    _tcscpy(pVer->lpszExtensionDesc, sz);
    return TRUE;
}
```

Для запуска этого модуля целесообразно использовать HTML-документ, содержащий следующий текст:

```
<A href="http://localhost/scripts/my.dll/name1">
Загрузить ISAPI-модуль по запросу http://localhost/Scripts/ my.dll/
name1</A>
<A href="http://localhost/scripts/my.dll/name2">
Загрузить ISAPI -модуль по запросу http://localhost/Scripts/my.dll/
name2</A>
```

В приведенном выше HTML-документе запросы отличаются параметрами “/name2” и “/name1”. При выборе одной из ссылок ISAPI-модуль будет вызываться для генерации соответствующего HTML-документа.

В ISAPI-модуле при выборе первой ссылки будет выполняться соответствующая команда на языке Transact-SQL. Выбор требуемого действия реализуется с помощью условного оператора, в котором переменная **bsForCommand** инициируется с использованием функции **AllocSysString()**, принадлежащей переменной **csForCommand** или **csForCommand1** в зависимости от значения переменной **par**:

```
CString csForCommand("select * from Authors where author='Simpson,  
Alan'");  
CString csForCommand1("select * from Authors where author='Jones,  
Edward'");  
if (!strcmp(par,"/name1"))  
    bsForCommand= csForCommand.AllocSysString();  
else  
    bsForCommand= csForCommand1.AllocSysString();
```

При этом переменная **par** определяется следующим образом:

```
char par[10];  
dwSize = 10;  
pCtxt->m_pECB->GetServerVariable(pCtxt->m_pECB->ConnID,  
(LPSTR)"PATH_INFO", (LPVOID)par, &dwSize);  
strcat(resBuff, par);
```

Здесь функция **GetServerVariable()** используется для получения значения системной переменной PATH_INFO, которую сервер помещает текстовую строку, содержащую параметры URL-запроса, находящиеся после знака “/”.

В переменной **m_pECB** содержится указатель на объект типа EXTENSION_CONTROL_BLOCK. Функция **strcat(resBuff, par)** выполняет конкатенацию строк, при этом результат конкатенации будет помещен по адресу, находящемуся в первом аргументе. В переменной **pCtxt** находится указатель на контекст вывода; в выражении ***pCtxt << _T(par)**; используется переопределенный оператор, пересылающий строку символов в контекст вывода.

В результате работы этого модуля в окно обозревателя при выборе ссылки <http://localhost/scripts/my.dll/name1> будет загружен HTML-документ, вид которого приведен на рис. 16.7.

При выборе второй ссылки в окне обозревателя будет находиться текст, аналогичный приведенному на рис. 16.7, за исключением строки “Значение переменной PATH_INFO:/name2” и записи в таблице – “Jones, Edward 1952”.

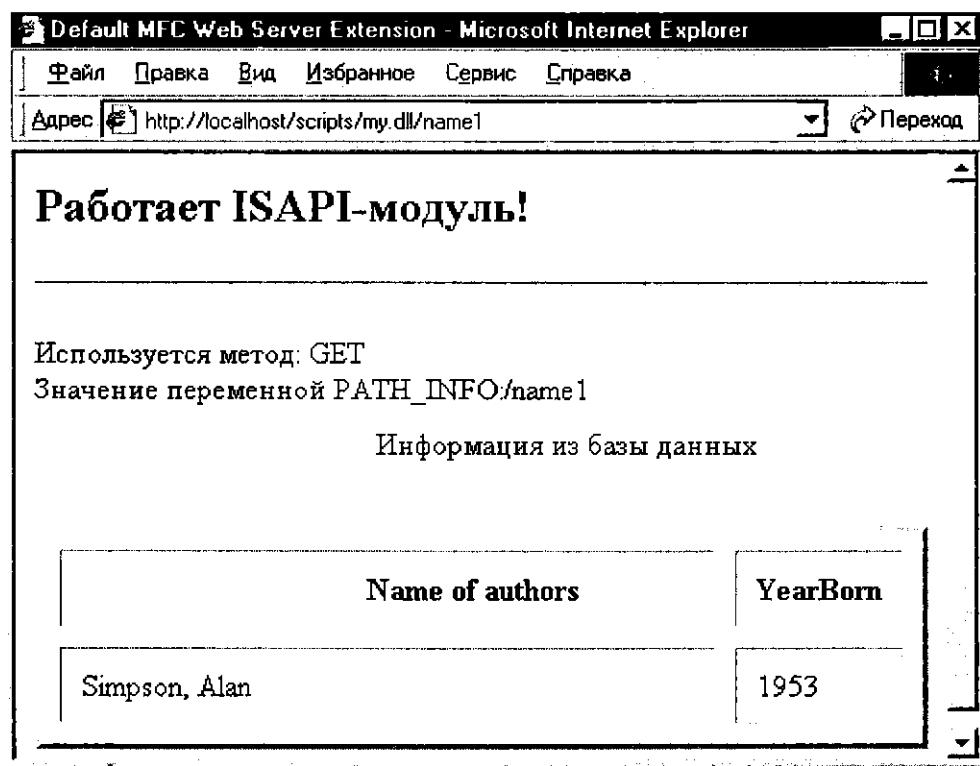


Рис. 16.7. Вид окна обозревателя с результатом работы модуля ISAPI

Контрольные вопросы и задания

1. Охарактеризуйте общий интерфейс взаимодействия CGI.
2. Опишите методы POST и GET передачи параметров запроса.
3. Приведите схему взаимодействия сервера и CGI-модуля.
4. Укажите данные, которые содержатся в переменных окружения.
5. Что определяет тип MIME?
6. Что представляет собой интерфейс ODBC и каким образом он может использоваться при создании Web-приложений?
7. Какие операции, установленные интерфейсом ODBC, должны быть выполнены для получения доступа к БД?
8. Запишите оператор обращения к функции SQLConnect() для установки соединения с источником данных с использованием имени DSN.
9. Как после установления соединения задаются запросы по извлечению данных с помощью команд языка SQL?

10. Охарактеризуйте интерфейс ADO и приведите иерархию его объектов.
11. Как выполняется непосредственно соединение с источником данных при использовании интерфейса ADO?
12. Дайте общую характеристику интерфейсу программирования серверных приложений ISAPI.
13. Укажите назначение функции `HttpExtensionProc()` и приведите ее прототип.
14. Назовите классы, используемые при разработке ISAPI-модуля.
15. Как выполняется получение переданных от обозревателя данных ISAPI-модулем?
16. Каким образом выполняется отправка данных ISAPI-модулем?
17. Выполнить разработку CGI-модуля, позволяющего шифровать передаваемые данные при использовании метода GET. Для шифрования можно применять произвольный алгоритм.
18. Выполнить разработку CGI-модуля, позволяющего шифровать передаваемые данные при использовании метода POST. Для шифрования можно применять произвольный алгоритм.

Литература

1. *Дунаев С. Б. Технологии Интернет-программирования.* СПб.: БХВ-Петербург, 2001.
2. *Мещеряков Е. В., Хомоненко А. Д. Публикация баз данных в Интернете.* СПб.: БХВ-Петербург, 2001.
3. *Фролов А. В., Фролов Г. В. Базы данных в Интернете: практическое руководство по созданию Web-приложений с базами данных.* М.: Издательско-торговый дом «Русская редакция», 2000.

17. Публикация БД с использованием XML

В главе рассматривается технология применения XML в качестве стандартного средства организации обмена данными при публикации БД в Интернете. Приводятся примеры программ, выполняющих передачу данных из базы данных в документ XML и обратно.

17.1. XML как средство обмена данными

Как отмечалось, расширяемый язык разметки XML (eXtensible Markup Language) представляет собой развитие языка HTML и по сравнению с ним обеспечивает ряд дополнительных возможностей. Главное отличие XML от HTML заключается в том, что с его помощью выполняется не только наполнение создаваемого документа содержанием с указанием разметки, а в основном определяется структура документа и типы хранимых в нем данных.

Напомним, что одно из достоинств XML состоит в том, что в разрабатываемых с его помощью документах описание структуры хранимых данных отделено от собственно данных. В связи с этим XML представляет собой удобное средство обмена данными между различными приложениями. Он позволяет обеспечить согласованный обмен данными между приложениями, в которых отличаются структура хранимых данных, например имена и типы полей.

Кроме того, с помощью XML мы можем упростить доступ к данным, хранимым в базах данных различных СУБД. Например, для доступа к данным персональных СУБД или табличного процессора Excel пользователю требуется установка соответствующих инструментальных средств. В этом случае можно создать активные серверные страницы (ASP) или сценарии на языке JScript или VBScript, которые выполняют извлечение данных из базы данных и помещение их в документ XML.

Информацию из полученного таким образом документа XML можно легко использовать в других приложениях или отображать на страницах HTML. Таким образом, полученные в результате данные становятся доступными для всех пользователей, имеющих обозреватель, независимо от наличия или отсутствия СУБД или табличного процессора.

Документы XML могут использоваться на стороне клиента и на стороне сервера. Возможности работы с документами XML на стороне *клиента* в настоящее время сдерживаются в основном отсутствием соответствующих инструментальных средств. Не все обозреватели предоставляют возможность

просмотра и работы с документами XML, но в современных продуктах такая возможность уже имеется. В частности, возможность просмотра XML-документов обеспечивается Internet Explorer, начиная с версии 4.0, а также Netscape Navigator 5.0 и выше.

При работе с документами XML на *стороне сервера* привлекаются языковые средства, обычно применяемые для расширения возможностей сервера, такие как Java и JScript. Отметим также, что последние версии современных систем программирования, ориентированных на разработку приложений Web, также имеют средства поддержки создания и обработки документов XML. В частности, соответствующие средства имеются в составе JBuilder 4.0.

В качестве причин и достоинств обработки данных XML-документа на сервере можно отметить следующее:

- обработка данных на сервере обязательно требуется при запуске кода, размещенного на сервере. В частности, при извлечении информации из базы данных;
- во вторых, обработка данных на сервере позволяет устраниить необходимость использования обозревателей, например Internet Explorer 5.0, поддерживающих обработку XML-документов. Можно предположить, что со временем все большее число обозревателей будет поддерживать такую обработку.

Один из недостатков обработки данных на сервере связан с тем, что при необходимости внесения каких-либо изменений в формируемом документе клиента ему требуется обращаться за помощью к серверу, создавая тем самым дополнительную нагрузку.

При выполнении обработки данных на *стороне клиента* необходимость обращения к серверу пропадает и можно выполнять требуемые изменения с помощью сценариев или динамического HTML.

17.2. Создание и обработка XML-документов

Для создания и обработки XML-документов используются разнообразные инструментальные средства. В частности, для создания документа XML и соответствующего ему файла DTD определения типа документа можно воспользоваться редактором XML. Для этой цели можно также воспользоваться любым текстовым редактором или процессором, например WordPad или Word. В последних случаях нужно иметь в виду, что файл XML является текстовым файлом с расширением xml и при сохранении его нужно задавать именно текстовый формат.

Создание XML-документов можно выполнять также путем преобразования уже имеющихся документов других приложений. Очевидно, что проще

всего преобразовать в XML-документ HTML-страницы. При этом нужно обеспечить дополнительные ограничения, накладываемые со стороны XML. Можно получить также XML-документы путем преобразования следующих типов документов: файлов PDF (Portable Document Format), используемых для представления в Интернете текстовых и числовых данных большого объема; документов табличных процессоров, таких как Excel; файлов баз данных, например СУБД Access; файлов документов текстовых процессоров, таких как Word.

В общем случае преобразование в XML-документ из файла документ другого формата может представлять достаточно трудоемкую задачу. Для решения задачи такого преобразования могут быть задействованы разнообразные инструментальные средства, такие как редакторы XML, синтаксические обозреватели и процессоры, серверные продукты, обозреватели Web, инструментальные средства проверки ссылок и синтаксиса.

Обработка XML-документов обычно включает в себя синтаксический анализ и собственно обработку, например, для отображения документа обозревателем или для извлечения данных из документа и помещения их базу данных. Синтаксический анализ XML-документов может выполняться с помощью синтаксических анализаторов или с помощью встроенных средств современных обозревателей Web. В частности, в составе обозревателя Internet Explorer 5.0 имеется встроенное средство синтаксического анализа, называемое MSXML.

С помощью синтаксических анализаторов выполняется считывание XML-документа и построение соответствующего этому документу иерархического дерева разбора данных. С их помощью можно создавать также соответствующее представление XML-документа. Для синтаксического анализа XML-документов используются два основных подхода. Первый подход основан на использовании объектной модели документа DOM (Document Object Model), позволяющей отображать любые объекты в XML-документе. Второй подход представляет собой простой интерфейс прикладного программирования SAX (Simple API for XML).

При синтаксическом анализе с помощью объектной модели DOM выполняется считывание всего XML-документа, создание иерархического дерева разбора и возврат данных из этого дерева. Этот способ не позволяет пользователю вмешиваться в процесс анализа и в наибольшей степени подходит для выполнения синтаксического анализа документов не очень большого объема. Отметим, что описанный способ поддерживается с помощью анализатора MSXML в составе Internet Explorer 5.0.

Анализ XML-документа с помощью объектной модели DOM требует создания экземпляра анализатора. Для этого можно использовать объект ActiveX, представляющий собой документ. После этого можно выполнять

доступ к данным документа с помощью свойств и методов созданного объекта. При этом созданный объект-документ предварительно нужно связать с анализируемым XML-документом по его URL с помощью метода `load` объекта. После нужно указать корневой элемент документа и можно выполнять перемещение по структуре документа. Описанные действия могут быть реализованы с помощью следующих строк кода:

```
var vxmldoc = new ActiveXObject ("microsoft.xmldom");
vxmldoc.load("MyXMLDoc.xml");
var vroot = vxmldoc.documentElement;
```

Здесь в качестве аргумента метода `ActiveXObject` может быть задана также строка "msxml". Для доступа к дочерним элементам объекта-документа могут использоваться его свойства `children` или `childNodes`.

Второй подход SAX обеспечивает интерфейс пользователя с синтаксическим анализатором. При этом выполняется интерактивный разбор не всего XML-документа в целом, а представляющей в настоящее время наибольший интерес части. При этом пользователь может перемещаться по документу и информировать синтаксический анализатор об определенных событиях, например, выявлении начала следующего элемента. Этот подход позволяет выполнять синтаксический анализ больших XML-документов. Упоминавшийся нами анализатор MSXML в составе Internet Explorer 5.0 не поддерживает подход SAX, но в дополнение к нему поставляются разработки других фирм, позволяющие его использование.

17.3. Сценарий для отображения XML-документа

Как отмечалось, современные обозреватели, например, Internet Explorer версии 5.0, позволяют проверять правильность и просматривать XML-документы. При просмотре в обозревателях можно управлять степенью детальности отображения их иерархической структуры. Например, пусть имеется XML-документ, размещенный в файле `myNew1.xml` и содержащий следующий код:

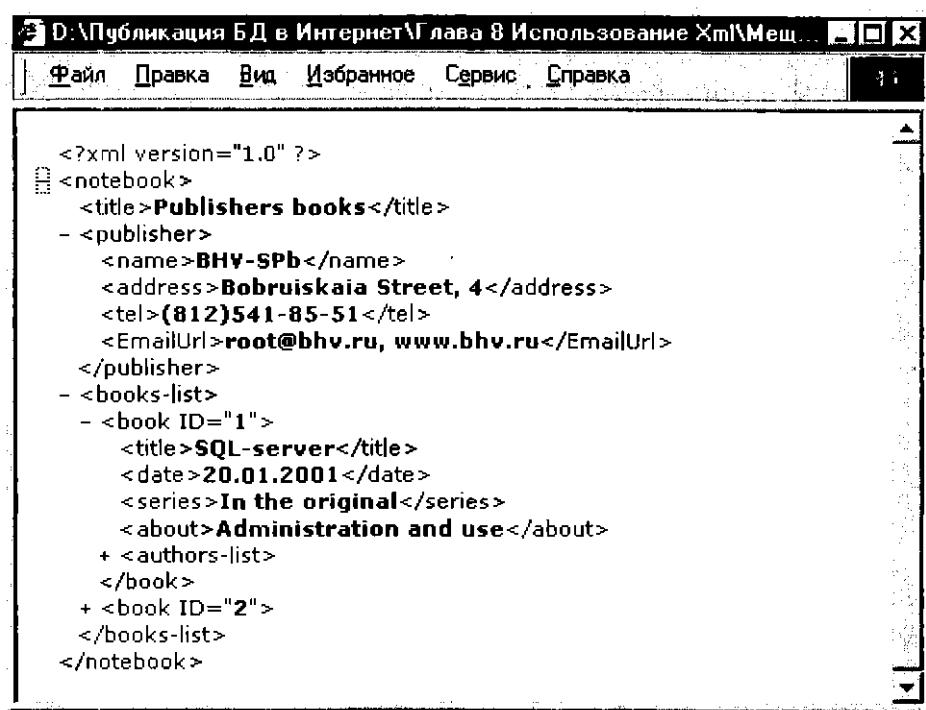
```
<?xml version="1.0"?>
<notebook>
<title>Publishers books</title>
<publisher>
<name>BHV-SPb</name>
<address>Bobruiska Street, 4</address>
<tel>(812)541-85-51</tel>
<EmailUrl>root@bhb.ru, www.bhb.ru</EmailUrl>
</publisher>
```

```
<books-list>
<book ID="1">
<title>SQL-server</title>
<date>20.01.2001</date>
<series>In the original</series>
<about>Administration and use</about>
<authors-list>
<author ID="1">
<firstname>Eugene</firstname>
<lastname>Mamaev</lastname>
</author>
</authors-list>
</book >
<book ID="2">
<title>Database publication in the Internet </title>
<date>21.03.2001</date>
<series>Master</series>
<about>Technologies of the publication in the Internet</about>
<authors-list>
<author ID="1">
<firstname>Eugene</firstname>
<lastname>Mescheriakov</lastname>
</author>
<author ID="2">
<firstname>Anatoly</firstname>
<lastname>Khomonenko</lastname>
</author>
</authors-list>
</book >
</books-list>
</notebook>
```

Как следует из текста, XML-документ служит для размещения информации о книгах издательств. В окне обозревателя приведенный документ может быть представлен в виде, показанном на рис. 17.1.

В ряде случаев может потребоваться отображение отдельных (или всех) компонентов XML-документа с заданием требуемого их расположения в окне обозревателя. Для этого можно написать клиентскую программу в виде Java-апплета или сценария JScript.

Рассмотрим решение задачи отображения в требуемом виде приведенного выше XML-документа в окне обозревателя с помощью сценария JScript и объектной модели DOM.



```
<?xml version="1.0" ?>
<notebook>
  <title>Publishers books</title>
  - <publisher>
    <name>BHV-SPb</name>
    <address>Bobruiskaya Street, 4</address>
    <tel>(812)541-85-51</tel>
    <EmailUrl>root@bhw.ru, www.bhw.ru</EmailUrl>
  </publisher>
  - <books-list>
    - <book ID="1">
      <title>SQL-server</title>
      <date>20.01.2001</date>
      <series>In the original</series>
      <about>Administration and use</about>
      + <authors-list>
      </book>
    + <book ID="2">
    </books-list>
  </notebook>
```

Рис. 17.1. Вид XML-документа notebook.xml в окне обозревателя

Документ HTML с решающим нашу задачу сценарием JScript содержит следующий код:

```
<HTML> <head> <title></title>
<script language="javascript">
var xmldoc = new ActiveXObject("msxml");
var xmlsrc = "myNew1.xml";
function viewTitle(elem){
// Отображение заголовка документа, определяемого элементом <title>
window.document.writeln('<center><table width="100%"'
border=0><tr><td width="100%" align="center" ><b><font'
color="black">' + elem.text + '</font></b></td></tr></table></'
center><br>'); }

function viewPublisherList(elem)
{ // Отображение содержимого дочерних элементов <Publisher>
var cur_item=elem.children.item("name");
window.document.writeln('<tr><td align="center" colspan="2"'
```

```
bgcolor="gray">><b><font color="white">Издательство
'+cur_item.text+'</font></b></td></tr>');
window.document.writeln('<tr><td bgcolor="silver"
colspan="2"><center><table width="80%" border=0>');
if(elem.type==0)
{ if(elem.children!=null)
{ window.document.writeln('<tr><td colspan=2 width="60%"> </td></tr>');
var cur_item=elem.children.item("address");
if(cur_item!=null)
{ window.document.writeln('<tr><td><font color="green">Адрес<
font></td><td align="right" ><b><font
color="gray">' +cur_item.text+'</font></b></td></tr>');
} var cur_item=elem.children.item("tel",0);
if(cur_item!=null)
{ window.document.writeln('<tr><td><font color="green">Телефон<
font></td><td align="right" ><b><font
color="gray">' +cur_item.text+'</font></b></td></tr>');
} var cur_item=elem.children.item("EmailUrl");
if(cur_item!=null)
{ window.document.writeln('<tr><td><font color="green">E-
Mail,URL</font></td><td align="right" ><b><font
color="gray">' +cur_item.text+'</font></b></td></tr>');
} var cur_item=elem.children.item("url");
if(cur_item!=null)
{ window.document.writeln('<tr><td><font color="green">URL<
font></td><td align="right" ><b><font
color="gray">' +cur_item.text+'</font></b></td></tr>');
} }
window.document.writeln('<tr><td colspan=2 width="60%"> </td></tr>');
window.document.writeln('</table></center></td></tr>');
}

function viewBooksList(elem)
{ // Отображение содержимого дочерних элементов <books-list>
window.document.writeln('<tr><td align="center" colspan="2"
bgcolor="gray"><b><font color="white">Книги издательства
</font></b></td></tr>');
window.document.writeln('<tr><td bgcolor="silver"
colspan="2"><center><table width="80%" border=0>');
if(elem.type==0)
{ if(elem.children!=null)
{ for(i=0;i<elem.children.length;i++)

```

```
{ var cur_book=elem.children.item("book",i);
window.document.writeln('<tr><td colspan=2 width="60%"> </td></tr>');
if(cur_book.children!=null)
{ var cur_item=cur_book.children.item("title");
if(cur_item!=null)
{ window.document.writeln('<tr><td align="left" colspan="2" ><b><font
color="black">'+cur_item.text+'</font></b></td></tr>');
}
var cur_item=cur_book.children.item("series");
if(cur_item!=null)
{ window.document.writeln('<tr><td><font color="green">Серия</
font></td><td align="right" ><b><font
color="gray">'+cur_item.text+'</font></b></td></tr>');
}
var cur_item=cur_book.children.item("authors-list");
if(cur_item!=null)
{ window.document.writeln('<tr><td><font color="green">Авторы</
font></td><td align="right" ><b><font
color="gray">'+cur_item.text+'</font></b></td></tr>');
}
window.document.writeln('<tr><td colspan=2 width="100%"> </td></tr>');
} }})
function viewError()
{ window.document.writeln('<center><hr>Error was detected'); }
function parse(root)
{ if(root==null) return; var i=0; var elem;
if(root.children!=null)
{
// Если вложенные элементы не определены,
//то свойство children будет
//установлено в null
window.document.writeln('<center><table width="80%"'
border=0><tr><td>');
// Перебор дочерних элементов
for(i=0;i<root.children.length;i++)
{ elem=root.children.item(i);
// Разбор подэлементов <title>
if(root.children.item(i).tagName=="TITLE"){ viewTitle(elem);
}
// Разбор подэлементов <publisher>
if(elem.tagName=="PUBLISHER"){ viewPublisherList(elem);
}
// Разбор подэлементов <books-list>
```

```
if(elem.tagName=="BOOKS-LIST"){ viewBooksList(elem);
}
}
window.document.writeln('</td></tr></table>');
}

function viewDocument()
{// Загрузка XML документа
window.document.writeln('<body bgcolor="white">');
parse(xmlDoc.root);
// Начало разбора документа
window.document.writeln('</body>');
}
//<script language="javascript">
xmlDoc.URL = xmlsrc;
viewDocument();
//-->
</script>
</head>
```

Приведенный сценарий выполняет разбор содержимого XML-документа из файла notebook.xml и задает его отображение в окне обозревателя Internet Explorer 5.0, как показано на рис. 17.2.

Таким образом, в рассмотренном примере состав и тип извлекаемых для отображения данных определяются документом XML, а состав отображаемых данных и порядок их расположения в окне обозревателя определяются с помощью сценария JScript. Определенным недостатком рассмотренного решения является ориентация сценария JScript строго на обозреватель Internet Explorer.

17.4. Формирование XML-документа на основе базы данных

Одной из наиболее важных прикладных задач, связанных с публикацией баз данных в Интернете с помощью XML, является формирование XML-документов на основе данных из баз данных. Одним из вариантов решения названной задачи на *стороне клиента* является использование Java-апплета.

HTML-документ, содержащий вызов Java-апплета, формирующего XML-документ, содержит следующий код:

```
<HTML>
<HEAD>
```

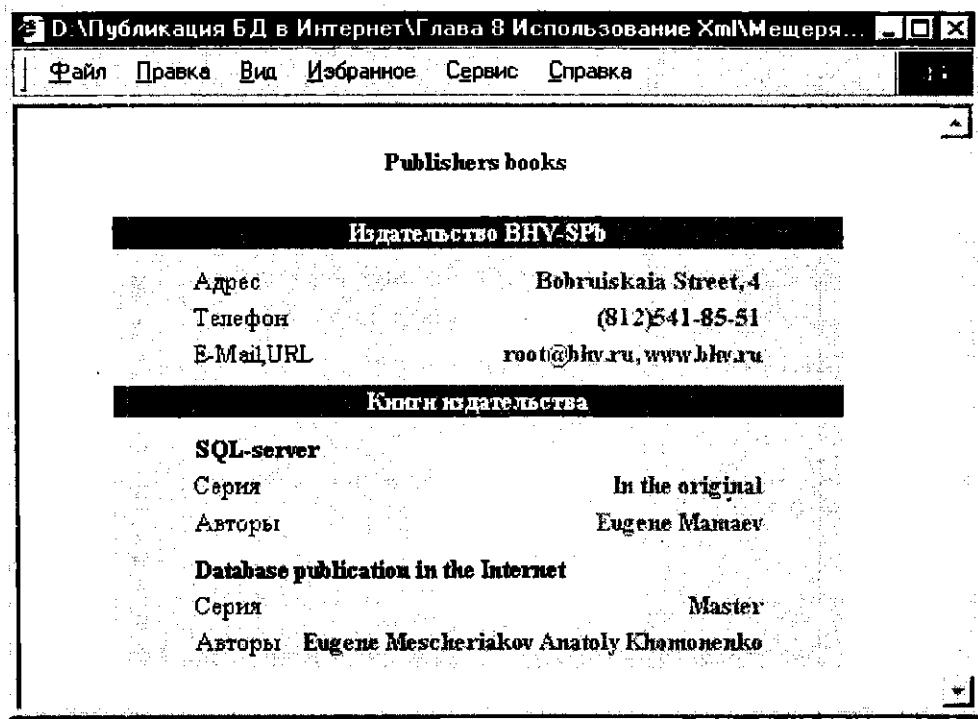


Рис. 17.2. Отображение XML-документа в окне обозревателя

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=windows-1251">
<TITLE>
HTML Test Page
</TITLE>
</HEAD>
<BODY>
untitled3.Applet1 will appear below in a Java enabled browser.<BR>
<APPLET
CODEBASE = "."
CODE = " AppletXML.Applet1.class"
NAME = "TestApplet"
WIDTH = 400
HEIGHT = 300
HSPACE = 0
VSPACE = 0
```

```
ALIGN = middle
>
</APPLET>
</BODY>
</HTML>
```

Апплет, выполняющий формирование XML-документа на основе данных из базы данных Authors, входящей в состав учебника по ASP-страницам в комплекте Windows 2000 Server, содержит следующий код:

```
package AppletXML;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import java.applet.Applet;
import java.awt.Graphics;
import java.util.Vector;
import java.sql.*;
import java.io.*;

public class Applet1 extends JApplet implements Runnable {
    private Thread worker;
    private Vector queryResults;
    private String message = "Инициализация";

    public synchronized void start() {
        // Функция "start" вызывается каждый раз при создании потока Thread
        if (worker == null) {
            message = "Соединение с БД";
            worker = new Thread(this);
            worker.start();
        }
    }

    //Функция run вызывается из методов объекта Thread.
    public void run() {
        // Задание строки URL-адреса источника БД
        String url = "jdbc:odbc:Authors";
        // Задание строки запроса
        String query = "Select * from Authors";
        try {
            //Инициализация драйвера
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
    } catch(Exception ex) {
        //Блок обработки ошибок инициализации драйвера
        setError("Can't find Database driver class: " + ex);
        return;
    }
    try {
        //Установка соединения с БД
        Connection con = DriverManager.getConnection(url, "", "");
        //Выполнение запроса к БД
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        //Определение файла myNew.xml для размещения
        //данных выходного потока
        try{
            FileOutputStream fil = new FileOutputStream("myNew.xml");
            DataOutputStream dos = new DataOutputStream(fil);
            PrintStream pos = new PrintStream (dos);
            pos.println("<?xml version='1.0'?>");
            pos.println("<notebook>");
            pos.println("<AuthorList>");
            int i=1;
            while (rs.next()) {
                //Из набора данных извлекается содержимое полей Author и
                //YearBorn из текущей записи
                pos.println("<Author ID='"+i+"'>");
                String s = rs.getString("Author");
                pos.println("<Name>");
                pos.println(s);
                pos.println("</Name>");
                String f = rs.getString("YearBorn");
                pos.println("<YearBorn>");
                pos.println(f);
                pos.println("</YearBorn>");
                pos.println("</Author>");
                i++;
            }
            pos.println("</AuthorList>");
            pos.println("</notebook>");
            pos.close();
        }catch(Exception ex){
            //Блок обработки ошибок выполнения операций вывода в файл
            setError("FileOutPutException: " + ex);
        }
    }
}
```

```
        }
        stmt.close();
        con.close();
    } catch(SQLException ex) {
        //Блок обработки ошибок выполнения SQL-запроса
        setError("SQLException: " + ex);
    }
}

//Функция paint вызывается для рисования внешнего вида апплета
//отображения содержимого
public synchronized void paint(Graphics g) {
    //Отображение строки message в случае если в результате выполнения
    //запроса возвращается пустой набор данных
    g.drawString(message, 5, 50);
    return;
}
// Функция используется для вывода сообщения об ошибке
private synchronized void setError(String mess) {
    message = mess;
    worker = null;
    // Вызов функции перерисовки апплета
    repaint();
}
}
```

Сформированный XML документ размещается в файле с именем myNew.xml. Вид этого документа в окне обозревателя Internet Explorer приведен на рис. 17.3.

Напомним, что достоинством использования Java-апплетов является высокая скорость их выполнения, поскольку апплеты хранятся в откомпилированном виде.

17.5. Размещение данных из XML-документа в базе данных

Размещение данных из XML-документа в базе данных некоторой СУБД представляет собой задачу, обратную по отношению к рассмотренной в предыдущем пункте. В этом случае требуется выполнить разбор XML-документа, выделить в нем требуемые данные, установить соединение ODBC с заранее созданной базой данных и поместить в нее выделенные данные с помощью SQL-запроса.

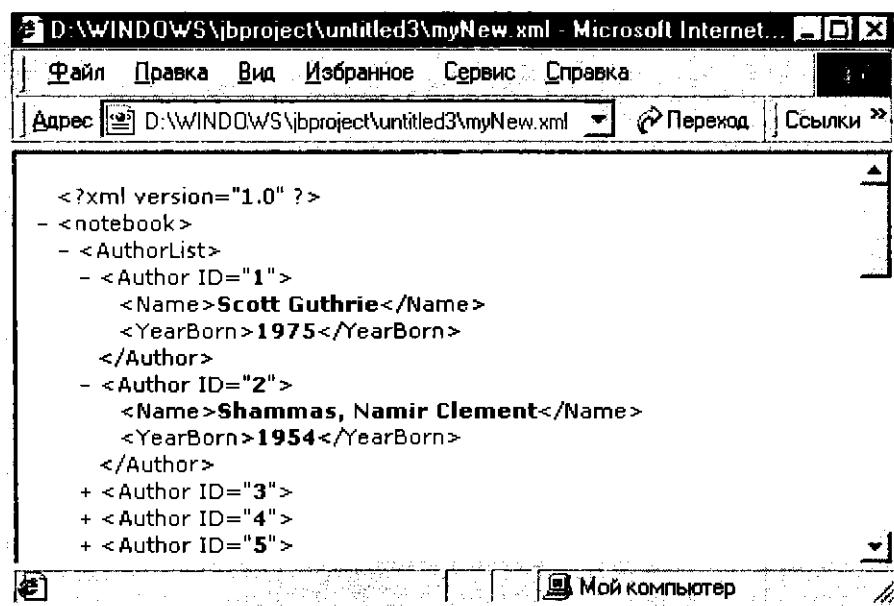


Рис. 17.3. Вид XML-документа в окне обозревателя

Рассмотрим пример решения указанной задачи при следующих предположениях. Имеется XML-документ, вид которого в окне Internet Explorer приведен на рис. 17.4. Требуется выбрать данные о книгах (название – book и авторы – authors) из документа и поместить в таблицу базы данных Microsoft Access и в текстовый файл.

В качестве варианта решения рассмотрим создание приложения JAVA с использованием объектной модели документа DOM. В состав проекта приложения JAVA входят два пакета (package) whritedb2 из файлов ReadXML2.java и MyClass.java. В первом файле содержится исходный код вида:

```
package whritedb2;
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.io.*;
import java.net.*;
import java.sql.*;
import java.util.*;
public class ReadXML2 {
    Connection con;
```

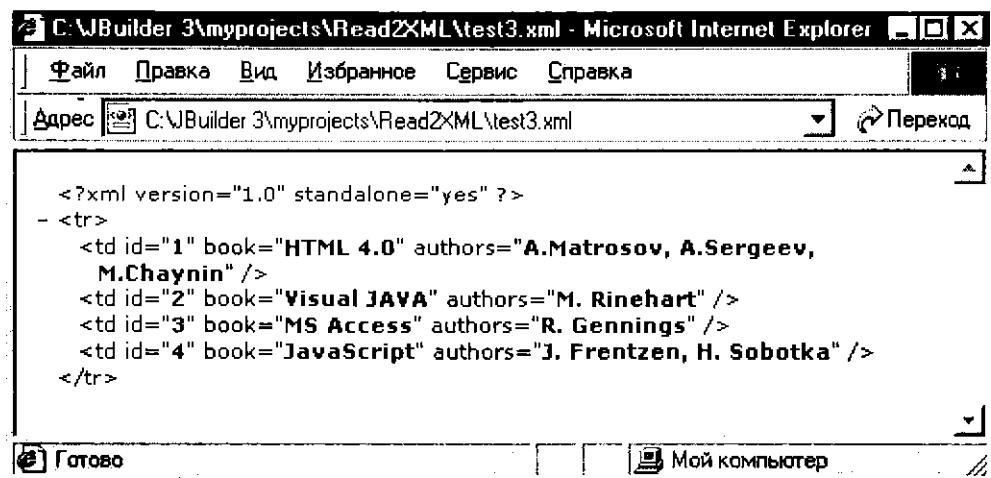


Рис. 17.4. Вид XML-документа в окне обозревателя

```
PreparedStatement stmt;
int count;
public ReadXML2() {
String url="jdbc:odbc:AccTab";
try { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
catch(java.lang.ClassNotFoundException e) {
System.err.print("ClassNotFoundException: ");
System.err.println(e.getMessage());
try {
con = DriverManager.getConnection(url);
// Подготовка шаблона для добавления записи в базу данных
stmt = con.prepareStatement("insert into AccessTab (w1,w2) val-
ues(?,?)");
} catch(SQLException ex) {
System.err.println("SQLException: " + ex.getMessage());
}
}
public void convertXMLDataToString(String aXMLFile, String aSaveFile){
DOMParser parser = new DOMParser();
// Тестирование XML-документа не выполняется
parser.setValidationMode(false);
// Выходной поток
```

```
PrintStream ps = null;
byte[] b = new byte[1000];
try {
    ps = new PrintStream(new FileOutputStream(aSaveFile));
}
catch (IOException e) {
    System.err.println("Exception: " + e.getMessage());
}

// Разбор XML-документа
try {
    parser.parse(fileToURL(aXMLFile));
}
catch (java.io.IOException ex){
    System.out.println("Exception in unParser (String message) –
java.io.IOException is " + ex.getMessage());
}
catch (org.xml.sax.SAXException ex){
    System.out.println("Exception in unParser(String message) –
org.xml.sax.SAXException is " + ex.getMessage());
}

// Получение данных от парсера, печать и сохранение
XMLDocument doc = parser.getDocument();

Element el = doc.getDocumentElement();

// Получение списка узлов с именем TD
NodeList nodeListTR = el.getElementsByTagName("td");
System.out.println("count = " + nodeListTR.getLength());
count = nodeListTR.getLength();
// Перебор всех узлов TD
for (int n = 0; n < nodeListTR.getLength(); n++) {

    Node attr = nodeListTR.item(n);
    NamedNodeMap nnm = attr.getAttributes();

    // Получение данных с именем book
    Node nBook = nnm.getNamedItem("book");
    // Получение данных с именем authors
    Node nAuthors = nnm.getNamedItem("authors");

    // Получение значений данных с именем book
    String s1 = (String)nBook.getNodeValue();
    //s1.addElement((String)nBook.getNodeValue());
```

```
// Получение значений данных с именем authors
String s2 = nAuthors.getNodeValue();
//s2.addElement((String)nAuthors.getNodeValue());
try {
    stmt.setString(1,s1);
    stmt.setString(2,s2);
    // Добавление записи в базу данных
    int rs = stmt.executeUpdate();
} catch(SQLException ex) {}

String data = s1 + "\r\n" + s2;
System.out.println(data);
data = data + "\r\n";
b = data.getBytes();

try {
    // Запись данных в выходной текстовый файл
    ps.write(b);
}
catch (IOException e){
    System.err.println("Exception: " + e.getMessage());
}
}
ps.close();
}

// Преобразование имени файла в URL-формат
private URL fileToURL(String aXMLFile) {
    File file = new File(aXMLFile);
    String path = file.getAbsolutePath();
    String fSep = System.getProperty("file.separator");
    if ((fSep != null) && (fSep.length() == 1))
        path = path.replace(fSep.charAt(0), '/');
    if ((path.length() > 0) && (path.charAt(0) != '/'))
        path = '/' + path;
    try {
        return new URL("file", null, path);
    }
    catch (java.net.MalformedURLException e) {
        // Исключение возникает, если файловый протокол не распознан
        throw new Error("unexpected MalformedURLException");
    }
}
```

В файле MyClass.java. содержится следующий исходный код:

```
package whritedb2;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import java.util.*;

public class MyClass1 {
    Connection con;
    PreparedStatement stmt;
    public MyClass1()
    {
        ReadXML2 read = new ReadXML2();
        read.convertXMLDataToString("test3.xml", "test3.txt");
    }

    public static void main(String[] args) {
        MyClass1 myClass1 = new MyClass1();
        myClass1.invokedStandalone = true;
    }
    private boolean invokedStandalone = false;
}
```

С помощью класса **ReadXML2** выполняется получение данных из XML-файла и запись их в таблицу **AccessTab** базы данных Microsoft Access **dbook.mdb** и в текстовый файл, при этом экземпляр названного класса не создается, а вызывается статическим методом **convertXMLDataToString**. Добавление записей в таблицу выполняется с помощью строки кода **int rs = stmt.executeUpdate();**.

В классе **MyClass1** выполняется вызов метода **convertXMLDataToString()** объекта **read** класса **ReadXML2**, с помощью которого и выполняется решение нашей задачи. Создание объекта **myClass1** класса **MyClass1** выполняется в классе **main**. Вид выбранных из XML-документа текстовых данных в окне редактора **WordPad** приведен на рис. 17.5.

Применительно к локальной базе данных **Access** для размещения в ней выбранных из XML-документа данных требуется установить псевдоним соединения с использованием Менеджера ODBC через панель управления Windows. С помощью псевдонима (в нашем случае **AccTab**) в приложении Java указывается база данных, с которой установлено соединение. В примере для этого используется строка кода **String url= "jdbc:odbc:AccTab";**.

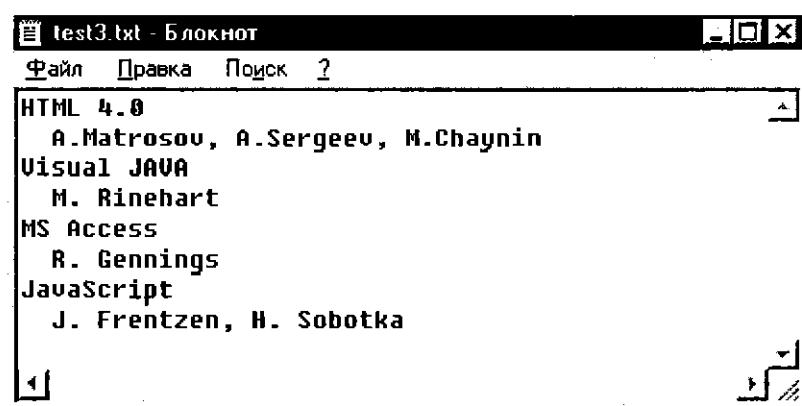


Рис. 17.5. Вид сформированных текстовых данных в окне редактора

Контрольные вопросы и задания

1. Укажите причины, достоинства и недостатки обработки данных XML-документа на сервере.
2. Назовите способы и средства, используемые для создания и обработки XML-документов.
3. Охарактеризуйте основные подходы, используемые для синтаксического анализа XML-документов.
4. Опишите работу приведенного во втором разделе главы сценария на языке JScript для отображения XML-документа в окне обозревателя.
5. Охарактеризуйте состав HTML-документа, содержащего вызов Java-апплета, формирующего или обрабатывающего XML-документ.
6. Составьте сценарий на языке JScript для отображения созданного вами XML-документа в окне обозревателя с использованием объектной модели DOM.
7. Разработайте Java-апплет, выполняющий формирование XML-документа на основе данных из произвольной базы данных СУБД Access.
8. Разработайте Java-апплет, выполняющий размещение данных из XML-документа в базе данных СУБД Access.

Литература

1. *Вебер Дж.* Технология JavaФ в подлиннике: Пер. с англ. СПб.: ВНВ –Санкт-Петербург, 1999.
2. *Вейнер П.* Языки программирования Java и JavaScript: Пер. с англ. ЛОРИ, 1998.
3. *Матросов А. В., Сергеев А. О., Чайнин М. П.* HTML 4.0. СПб.: БХВ – Санкт-Петербург, 2000.

4. Коннолли Т., Бэгг К. Базы данных. Проектирование. Реализация и сопровождение. Теория и практика. — М.: Вильямс, 2003.
5. Мещеряков Е. В., Хомоненко А. Д. Публикация баз данных в Интернете. — СПб.: БХВ-Санкт-Петербург, 2001.
6. Питс-Моултис Н., Кирк Ч. XML / Пер. с англ. -- СПб.: BHV — Санкт-Петербург, 1999.
7. Harold E. R. XML: Extensible Markup Language. — IDG Books Worldwide, 1998.

18. Публикация БД средствами Microsoft Access

18.1. Характеристика вариантов публикации

При публикации БД в Интернете с помощью Microsoft Access 2000 можно создавать следующие три разновидности Web-страниц:

- страницы доступа к данным;
- серверные страницы;
- статические файлы HTML.

Общие сведения о *страницах доступа к данным* приведены в подразделе 11.3. Здесь лишь добавим, что они создаются как объекты базы данных, используемые для перехода к расположению соответствующего файла HTML. Чтобы сделать страницы доступа к данным доступными в Интернете, следует опубликовать их в папках Web или на Web-сервере и установить базу данных Access (или SQL Server) доступной пользователям страницы. Для просмотра данных страницы и взаимодействия с ними достаточно один раз загрузить страницу с Web-сервера в Web-обозреватель.

Серверные страницы (ASP или IDC/HTX) создаются из таблиц, запросов и форм. Серверные страницы используются, когда информация в БД часто изменяется. При этом связь с БД организуется с помощью ODBC-интерфейса в случае IDC/HTX-страниц или ADO-интерфейса в случае ASP-страниц.

Полученные выходные файлы формата ASP или IDC/HTX обрабатываются Web-сервером по запросу обозревателя. При каждом открытии или обновлении файла ASP или HTX пользователем Web-обозревателя Web-сервер динамически создает файл HTML, а затем отправляет его Web-обозревателю.

Статические файлы HTML можно создавать из таблиц, запросов, форм и отчетов. В Web-обозревателе отчеты отображаются в формате отчета, а таблицы, запросы и формы отображаются в формате таблиц. Статические файлы HTML используются для Web-обозревателей, поддерживающих HTML версии 3.2 и выше, при нечастом изменении данных. Чтобы сделать статические файлы HTML доступными в Интернете, следует опубликовать их в папках Web или на Web-сервере.

Для просмотра данных с помощью Web-обозревателя достаточно один раз загрузить статический файл HTML с Web-сервера. Статические файлы HTML содержат данные, полученные во время публикации файлов. Со статическим файлом HTML не связан ни один источник данных ODBC, поэтому при изменении данных нужно вновь экспортить файлы для просмотра новых данных в обозревателе.

Серверные страницы (ASP или IDC/HTX) и статические файлы HTML создаются путем экспорта соответствующих объектов баз данных (таблиц, запросов, форм и отчетов), из которых они создаются. Делается это с помощью команды **Экспорт (Export)** меню **Файл (File)** после выбора нужного объекта базы данных.

18.2. Страницы доступа к данным

Основная часть страницы представляет собой основную часть макета страницы доступа к данным. Основную часть страницы доступа к данным с группировкой можно использовать для отображения информационного текста и разделов. По умолчанию положение текста, разделов и других элементов на основной части страницы является *относительным* — элементы располагаются один за другим в порядке следования их в источнике HTML. Положение элемента определяется предшествующим содержимым. При просмотре содержимое основной части упорядочивается по размеру окна Web-обозревателя.

Разделы используются для отображения текста, данных и панелей инструментов. Положение элементов в пределах раздела по умолчанию является *абсолютным* — фиксировано относительно верхнего и левого краев раздела. Элементы управления с абсолютным положением внутри раздела остаются на том же месте и при изменении размеров окна обозревателя.

В зависимости от целей создания можно выделить три *типа страниц доступа к данным*, используемых для выполнения следующих действий:

- создание отчетов в интерактивном режиме;
- ввод и редактирование данных;
- наглядное отображение и анализ данных.

Для построения страниц доступа к данным используются различные компоненты. Для удобства просмотра страниц доступа к данным записи на них могут *группироваться*. Рассмотрим компоненты, используемые при создании страниц доступа к данным, а также преимущества и технику группировки записей на страницах доступа к данным. Затем приведем краткую характеристику и особенности создания указанных типов страниц доступа к данным.

Компоненты страниц

Страница доступа к данным может включать различные компоненты в зависимости от цели ее создания. Поля и записи всегда входят в состав страницы доступа к данным, но представлены могут быть по-разному. К числу наиболее часто используемых компонентов страниц доступа к данным относятся следующие.

Текстовые поля или поля ввода служат для отображения существующих данных из базы данных или для ввода пользователем новых данных в базу данных.

Записи представляют собой наборы связанных свойств об одном хранящемся в базе данных элементе. Например, данные о товаре определенной марки типа Напитки, поставщика Мушим's, хранящемся на складе в бутылках емкостью 750 мл, в количестве 69 единиц и общей стоимостью 810 р. составляют запись.

Группы служат для объединения данных в наборы. Например, записи данных о сотрудниках на странице доступа к данным можно сгруппировать по должностям. Каждая должность составляет заголовок группы сотрудников (рис. 18.1).

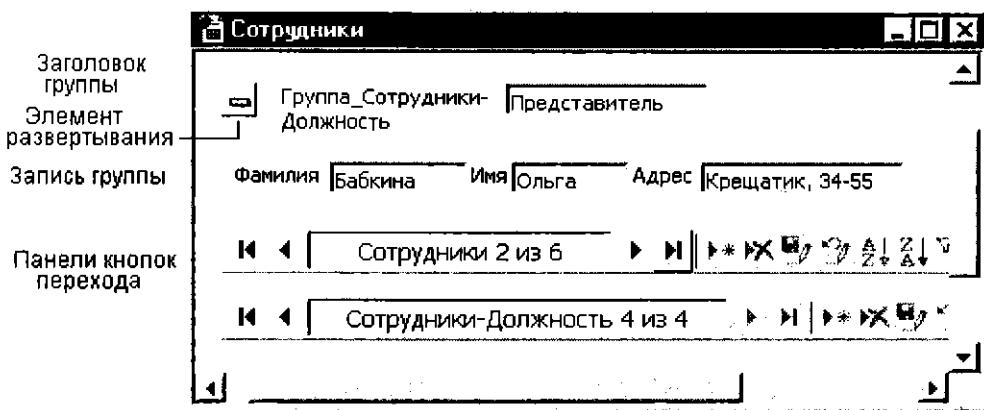


Рис. 18.1. Страница доступа с данными о сотрудниках

В зависимости от того, как разработана страница, на ней либо отображаются все группы (обычно со скрытием записей), либо ни одна конкретная группа не отображается до тех пор, пока не будет выбрана в списке доступных групп. Если на странице отображаются все группы, то можно отобразить все записи конкретной группы, щелкнув маркер развертывания. В нижней части каждой развернутой группы может находиться собственная панель перехода по записям. Вводить и изменять данные на странице с группами нельзя.

Панель кнопок перехода по записям (рис. 18.2) позволяет выполнять переходы между записями, добавлять, удалять, сохранять, отменять изменения, сор-

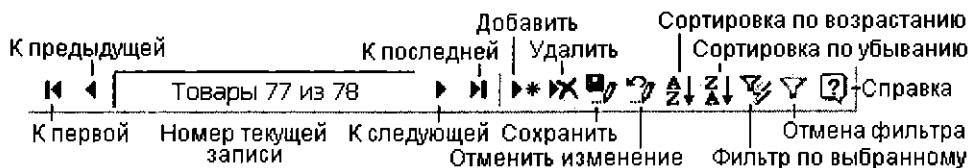


Рис. 18.2. Панель кнопок перехода по записям

тировать или фильтровать записи. На странице доступа группа может иметь собственную панель перехода по записям, расположенную в нижней части развернутой группы. Состав кнопок рассматриваемой панели можно настраивать с помощью команды **Настройка...** (*Customise...*) меню **Вид (View)**.

Сводные списки являются аналогом сводных таблиц Microsoft Excel и отображают данные в виде строк и столбцов, которые можно реорганизовывать для анализа данных различными способами (рис. 18.3). Это делается путем перемещения строк в столбцы и столбцов в строки, отображения итоговых сумм на пересечении строк и столбцов, а также с использованием других способов упорядочения и суммирования данных.

КварталИсполнения			
	Кв 1	Кв 2	Grand Total
Марка	Продажи Товаров	Продажи Товаров	Сумма продаж товаров
Chai	29127	17356,5	46483,5
Chang	13473,6	22362,7	35836,3
	13473,6	22362,7	
Chartreuse verte	14744,1	22720	37464,1

Рис. 18.3. Вид сводного списка

Управление сводным списком удобно выполнять с помощью его панели инструментов или команд контекстного меню.

Электронная таблица представляет собой аналог листа табличного процессора Microsoft Excel, позволяет вводить и редактировать данные и выполнять вычисления с ними. На рис. 18.4 приведен вид электронной таблицы расчета суммарной стоимости товара на складе.

Можно вводить данные в ячейки таблицы и с помощью формул задавать требуемые вычисления. В созданную таблицу можно также импортировать данные. После публикации листа на Web-странице пользователи обозревателя могут взаимодействовать с данными и выполнять вычисления непосредственно в обозревателе. Электронная таблица также имеет свою панель инструментов.

Диаграммы служат для визуального отображения численных данных из базы данных, тенденций и закономерностей их поведения (рис. 18.5). При изменении базы данных происходит соответствующее изменение диаграммы.

	B	C	D	E
1	Единица измерені	Цена	На складе	Стоимость
2	24 упаковки по 50	945,00р.	104	98280
3	12 упаковок по 25	405,00р.	61	24705
4	12 банок по 355 м	202,50р.	20	4050
5	20 банок по 450 г	630,00р.	76	47880
6	100 пакетов по 25	1 405,35р.	15	21080,25

Рис. 18.4. Вид электронной таблицы

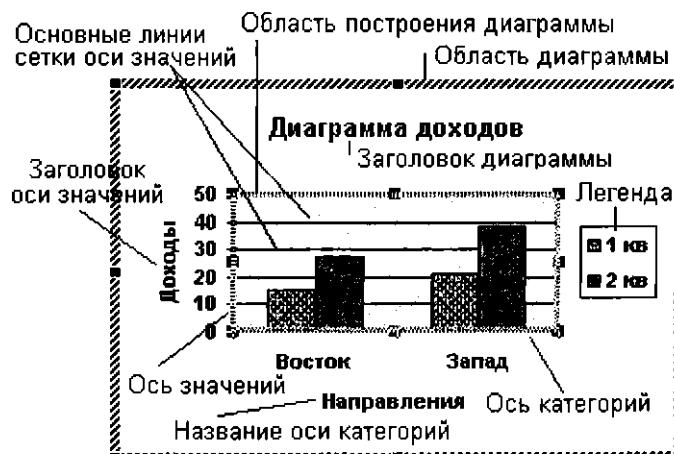


Рис. 18.5. Возможный вид плоской диаграммы

Диаграмма может быть связана со сводным списком или электронной таблицей на странице доступа к данным, при этом диаграмма изменяется при изменении связанного с ней сводного списка или электронной таблицы.

Группирование записей

Группирование записей на странице доступа к данным означает следующее: создается иерархия, которая группирует записи от общих категорий к конкретным деталям. На страницах доступа к данным с группировкой используются следующие четыре типа разделов — это раздел:

- заголовка группы — для отображения данных и вычисления итоговых значений. Для группировки данных нужны минимум два уровня группировки.

Как и раздел данных в отчете, заголовок группы на нижнем уровне повторяется до тех пор, пока не будут распечатаны все записи текущего раздела;

- примечаний группы — для вычисления итоговых значений. Он отображается перед разделом перехода по записям для уровня группировки. Примечание группы недоступно для нижнего уровня группировки;
- подписей — для отображения подписей столбцов данных. Раздел отображается непосредственно перед заголовком группы в том случае, когда развернута группа на следующем уровне сверху. В раздел подписей нельзя помещать присоединенные элементы управления;
- перехода по записям — используется для отображения кнопок перехода по записям для уровня группировки. Раздел отображается после раздела заголовка группы, если раздел подписей группы отсутствует, или после примечания группы, если оно имеется. В этот раздел нельзя помещать присоединенные элементы управления.

Замечание.

При добавлении уровней группировки на страницу доступа к данным содержимое ее изменять нельзя, его можно только просматривать.

Каждому уровню группировки соответствует свой *источник записей*. Имя источника записей отображается в строке заголовка каждого раздела для данного уровня группировки. На рис. 18.6 показана страница доступа к данным "Сотрудники" в режиме Конструктора. Внешний вид разделов на той же странице доступа к данным, открытой в режиме страницы или в Internet Explorer, показан на рис. 18.1.

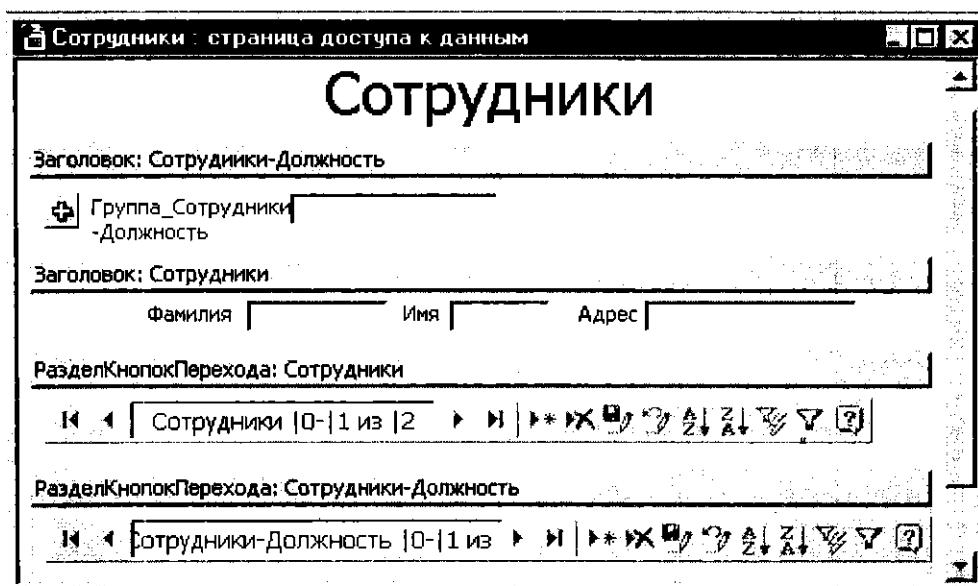


Рис. 18.6. Вид страницы доступа к данным в режиме Конструктора

Число записей, которые требуется отображать на странице для каждого уровня группировки, можно задать значением свойства **Размер страницы доступа к данным (DataPageSize)** в окне **Сортировка и группировка (Sorting and Grouping)** в режиме Конструктора страницы. В примере на рис. 18.1 свойство **Размер страницы доступа к данным (DataPageSize)** имеет значение 1 для уровня группировки “Сотрудники-Должность” и для уровня группировки “Сотрудники”.

Если для уровня группировки “Сотрудники” свойству **Размер страницы доступа к данным (DataPageSize)** установить значение 2, то рассматриваемая страница доступа к данным, открытая в режиме страницы или в Internet Explorer, примет вид, показанный на рис. 18.7.

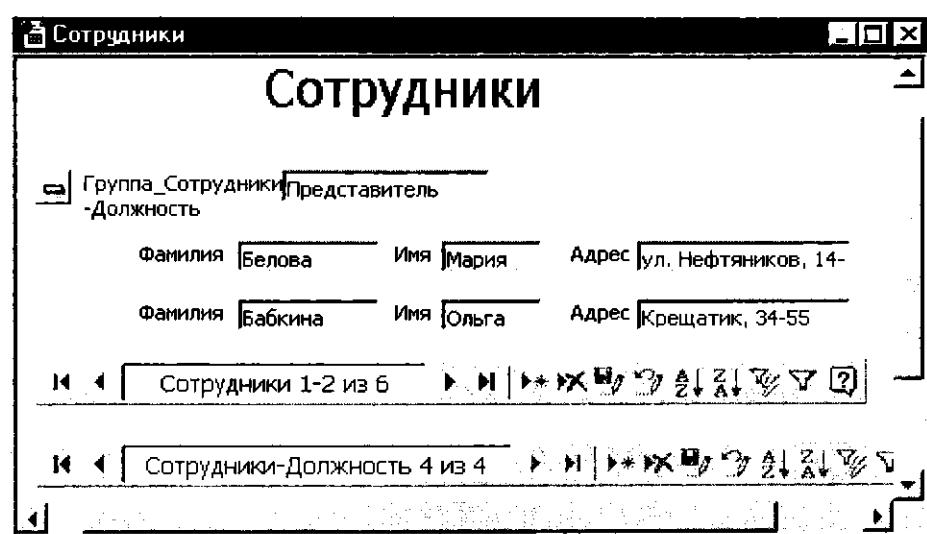


Рис. 18.7. Вид страницы доступа к данным в режиме Страницы

При создании страниц с *группированием* рекомендуется следующее:

- помещать присоединенные элементы управления в раздел;
- для отображения записей на самом нижнем уровне группировки использовать отдельные элементы управления или сводную таблицу;
- удалять ненужные кнопки с панели перехода по записям;
- использовать присоединенные элементы управления HTML вместо полей;
- установить свойству **Развернуто по умолчанию (ExpandedByDefault)** всех уровней группировки значение **Нет (None)**;
- свойству **Размер страницы доступа (DataPageSize)** в окне **Сортировка и группировка (Sorting and Grouping)**, определяющему число отображаемых в группе записей, установить по возможности меньшее значение.

Последние три рекомендации направлены на сокращение времени загрузки страницы доступа к данным в обозреватель.

Создание отчетов в интерактивном режиме

Создание отчетов в интерактивном режиме не требует изменения данных, хранимых в базе данных. Поэтому при разработке страниц доступа к данным этого типа допускается группирование данных, обеспечивающее взаимодействие с большими объемами выбранных данных. Разворачивая и сворачивая группы записей, пользователь может просматривать требуемые данные.

Пример 1. Создание отчетов в интерактивном режиме в виде страниц доступа к данным.

Предположим, что для учебной базы данных Борей, входящей состав СУБД Access 2000, требуется создать страницу доступа к данным таблиц **Доставка** и **Заказы**. Для решения поставленной задачи воспользуемся помощью Мастера страниц и Конструктора, выполнив следующую последовательность шагов.

- В среде СУБД откроем БД Борей, в подокне **Объекты (Objects)** выберем вариант **Страницы (Pages)**(рис. 18.8).

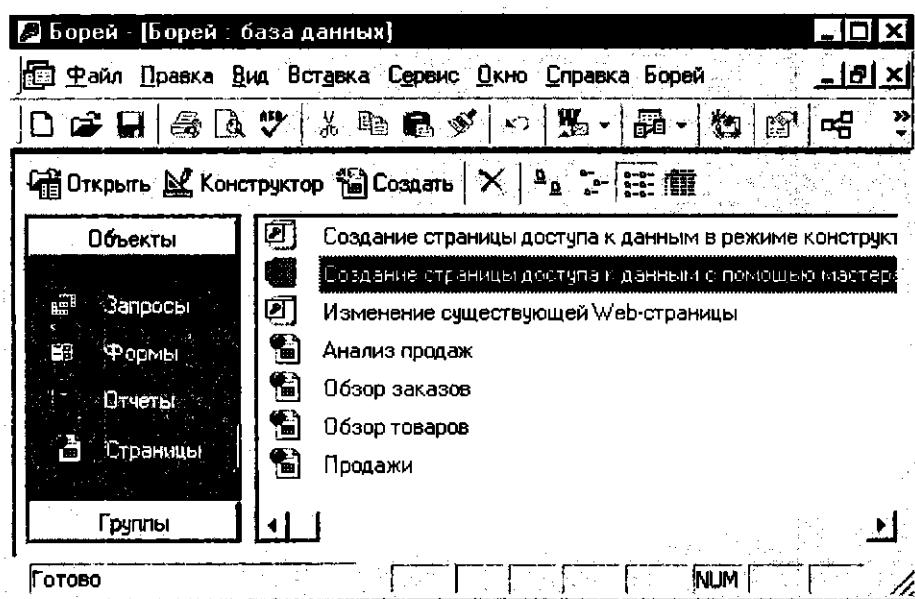


Рис. 18.8. Диалог выбора варианта создания страницы доступа к данным

- В поле справа выберем создание страницы с помощью Мастера. В открывшемся диалоговом окне поочередно выберем имена нужных таблиц (**Доставка** и **Заказы**), а также полей в них для вставки в страницу доступа к

данным. Нажмем кнопку **Далее (Next)**. При создании страниц доступа к БД Access допускается организовывать доступ к данным нескольких таблиц и запросов.

- В следующем диалоговом окне для удобства просмотра добавим уровень группировки по полю **КодДоставки**.
- В очередном диалоговом окне Мастера страниц для удобства просмотра данных страницы зададим сортировку по полю **Название**.
- В следующем окне зададим название страницы и выберем вариант изменения макета страницы для настройки и нажмем кнопку **Готово (Finish)**.
- В очередном окне выполним настройку макета созданной страницы, уточнив расположение элементов на странице доступа к данным подходящим образом. Закроем окно диалога, утвердительно ответим на вопрос о необходимости сохранения макета страницы.
- В очередном диалоговом окне укажем имя файла (**Страница_Доставка_-
Заказы**) для сохранения созданной страницы доступа к данным. При этом будет создан одноименный файл с расширением **html**.
- Выполним собственно публикацию страницы доступа к данным на Web-сервере. Для этого с помощью Проводника Windows скопируем соответствующий созданной странице файл HTML, а также все сопутствующие файлы (рисунки, таблицы стилей) и папки в папку корневого каталога Web-сервера. Стандартными корневыми каталогами являются **\Webshare\Wwwroot** для Personal Web Server и **\Inetpub\Wwwroot** для Microsoft Internet Information Server.

Созданную описанным способом страницу доступа к данным можно просматривать и настраивать в среду СУБД Access, а также просматривать с помощью обозревателя, обращаясь к странице на сервере. Вид указанной страницы в окне обозревателя Internet Explorer показан на (рис. 18.9).

Напомним, что созданная нами страница доступа к данным при работе с ней не допускает изменения данных, хранимых в базе данных. Поэтому при ее разработке мы использовали группирование данных, повышающее удобство взаимодействия с большими объемами выбранных данных.

Если в базу данных Борей (точнее, в исходные таблицы), которую мы использовали при создании страницы доступа к данным, будут внесены изменения, то они не получат отображения в нашей странице. Для этого требуется повторное создание новой страницы доступа к данным по описанной схеме.

Ввод и редактирование данных

Использование страниц доступа к данным для ввода данных аналогично использованию форм ввода данных: допускается ввод, редактирование и удаление данных в базе данных. Но достоинством страницы доступа к данным является возможность использования ее за пределами базы данных Microsoft Access, обновляя данные через Интернет или интранет.

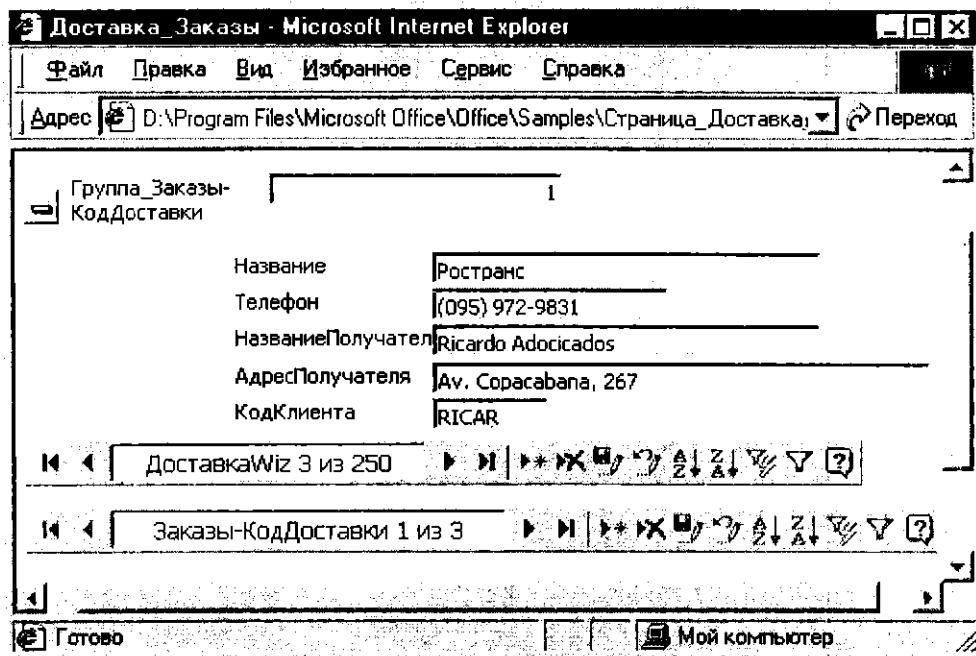


Рис. 18.9. Вид созданной страницы доступа к данным в окне обозревателя

При создании страницы для ввода данных нужно учитывать следующее:

- Используются отдельные элементы управления, такие как поля, списки, раскрывающиеся списки, группы параметров, переключатели и флагки.
- Элементы управления размещают в основной части страницы или в разделах. Не используемые разделы можно удалить.
- Создается только один уровень группировки. Окно **Сортировка и группировка (Sorting and Grouping)** содержит один источник группы записей.
- Свойству **Размер страницы доступа (DataPageSize)** в окне **Сортировка и группировка (Sorting and Grouping)** устанавливается значение 1.
- Создаются электронные таблицы, с помощью которых выполняются вычисления с полями записи. Вычисляемые значения можно отобразить в электронной таблице или скрыть таблицу и отобразить рассчитанные значения в присоединенных элементах управления HTML.
- Определяются способы использования данных пользователями путем настройки панели перехода по записям.
- Если требуется редактировать данные из таблиц, имеющих отношение «один-ко-многим», создают одну страницу, присоединенную к таблице на стороне «один» этого отношения, и вторую страницу, присоединенную к таблице на стороне «многие». Далее в диалоговом окне **Добавить**

гиперссылку (Add Hyperlink) можно установить связь между двумя страницами доступа к данным. Для обеспечения вывода данных с обеих страниц на одной странице с помощью редактора HTML создают кадры.

Наглядное отображение и анализ данных

Для наглядного отображения и анализа данных применяются диаграммы, электронные таблицы и сводные списки. С помощью *диаграмм* выполняется анализ тенденций, выявляются закономерности и производится сравнение данных в базе данных. Создание диаграмм выполняется с помощью компонента диаграммы.

Электронные таблицы позволяют вводить и редактировать данные и с помощью формул выполнять вычисления, допустимые в табличном процессоре Excel. При использовании электронных таблиц учитывается следующее:

- На странице доступа к данным *без группирования* электронную таблицу и другие элементы управления можно поместить в основную часть или в раздел.
- На странице доступа к данным *с группированием* электронная таблица и другие элементы управления, присоединенные к полям базы данных, помещаются в раздел. Электронные таблицы можно использовать на любом уровне группировки.
- Для раздела, содержащего электронную таблицу, свойству **Размер страницы доступа (DataPageSize)** в окне **Сортировка и группировка (Sorting and Grouping)** устанавливают значение 1.

Сводные таблицы позволяют организовывать данные различными способами. Сводную таблицу можно присоединить к данным базы данных или использовать данные с листа Microsoft Excel. Сводную таблицу можно использовать как единственный элемент управления на странице доступа к данным или вместе с другими элементами управления.

В зависимости от выполнения группирования данных на странице использование сводного списка осуществляется, как указывается ниже.

- На странице *без группирования* сводную таблицу и другие элементы управления можно поместить в основную часть или в раздел.
- При использовании сводного списка на странице доступа к данным *с группированием* требуется выполнить следующее:
 - поместить сводную таблицу и присоединенные элементы управления в раздел;
 - разместить сводную таблицу на самом нижнем уровне группирования;
 - удалить или скрыть раздел кнопок перехода для уровня группировки, на котором находится сводная таблица, если сводная таблица является единственным элементом управления в разделе ;
 - установить значение 1 свойству **Размер страницы доступа (DataPageSize)** в окне **Сортировка и группировка (Sorting and Grouping)** .

Пример 1. Создание сводного списка на странице доступа к данным. Рассмотрим создание сводного списка с помощью Конструктора на основе таблицы **Товары** базы данных Борей. Для решения поставленной задачи выполним следующую последовательность шагов.

- В среде Access откроем базу данных Борей, в левой части окна в списке объектов выберем вариант **Страницы (Pages)**(рис. 18.8) и зададим создание страницы доступа к данным с помощью Конструктора.
- В открывшемся диалоговом окне в поле с текстом «Название страницы» укажем название, например «Продажи по категориям», и с помощью кнопки **Сводная таблица Office (Pivot Table Office)** панели элементов выполним вставку объекта **Сводная таблица (Pivot Table)** в несвязанный раздел на странице доступа к данным.
- Зададим источник данных для сводного списка, для этого с помощью кнопки **Свойства (Properties)** панели **Макет страницы (Page Layout)** вызовем панель свойств для нашего объекта **Сводная таблица (Pivot Table)** и на его вкладке **Другие (Other)** (рис. 18.10) свойству **DataMember** присвоим путем выбором из списка значение **Товары** (имя таблицы-источника данных).

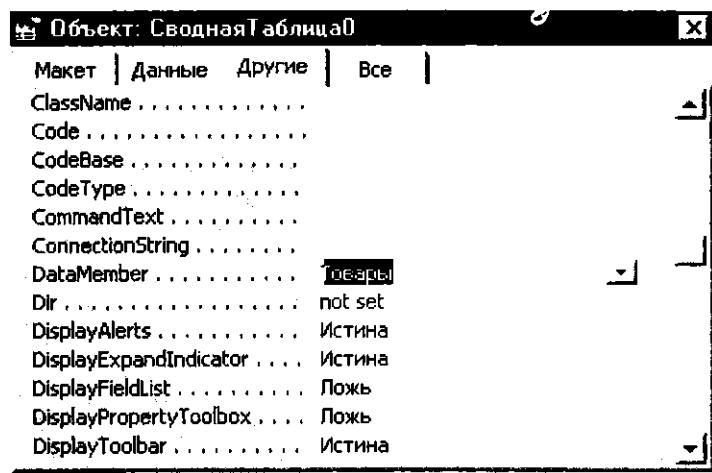


Рис. 18.10. Панель свойств объекта **Сводная таблица**

- Заполним поля сводной таблицы, для этого щелчком мыши выделим сводную таблицу, щелчком правой кнопкой мыши вызовем контекстное меню, зададим команду **Список полей (Field List)** и из открывшегося одноименного диалогового окна мышью перетащим требуемые поля в нужные места сводной таблицы.

- В очередном диалоговом окне укажем имя файла (Продажи_по_категориям) для сохранения созданной страницы доступа к данным.
- Выполним собственно публикацию страницы доступа к данным на Web-сервере. Для этого с помощью Проводника Windows скопируем соответствующий созданной странице файл HTML, а также все сопутствующие файлы (рисунки, таблицы стилей) и папки в папку корневого каталога Web-сервера.

Созданную описанным способом страницу доступа к данным можно просматривать и настраивать в среды СУБД Access, а также просматривать с помощью обозревателя, обращаясь к странице на сервере.

18.3. Серверные страницы

Серверные страницы представляют собой файлы ASP или IDC/HTX, которые создаются на основе таблиц, запросов или форм и подключаются к источнику данных ODBC или ADO соответственно. При их обработке с помощью Internet Information Server происходит динамическое создание файлов HTML. Отображаемые в полученных таким образом файлах HTML данные доступны только для чтения. Соответствующие им Web-страницы выводятся на экран в табличном формате в любом обозревателе Web.

Для экспорта таблицы, запроса или формы в генерируемый сервером формат HTML нужно выполнить следующие действия:

- В окне базы данных выбрать имя таблицы, запроса или формы, которую нужно экспорттировать, и в меню **Файл (File)** задать команду **Экспорт (Export)**.
- В открывшемся окне в поле **Тип файла (File Type)** выбрать требуемый вариант Microsoft IIS 1-2 (*.htx;*.idc) или Microsoft Active Server Pages (*.asp)(рис. 18.11), в зависимости от нужного генерируемого сервером формата.
- Выбрать диск и папку, в которую будет экспортирован объект.
- В поле **Имя файла (File Name)** ввести имя файла и нажать кнопку **Сохранить (Save)**.
- Ввести сведения в диалоговых окнах **Настройка вывода файлов ASP (Customize Output File ASP)** или **Настройка вывода файлов HTX и IDC (Customize Output File HTX/IDC)**(рис. 18.12).
- В поле **Шаблон HTML (Template HTML)** ввести местоположение шаблона HTML.
- В поле **Название источника (Data Source)** ввести имя источника данных ODBC, к которому будет производиться подключение при обработке генерируемых сервером файлов HTML на Web-сервере.
- В базе данных Microsoft Access указать в полях **вход под именем (Username)** и **пароль (Password)** имя пользователя и пароль системы защиты на уровне пользователей для разрешения доступа к базе данных

Microsoft Access с Web-страницы. По умолчанию будут действовать имя пользователя **Admin** и пустой пароль.



Рис. 18.11. Вид окна экспорта объекта

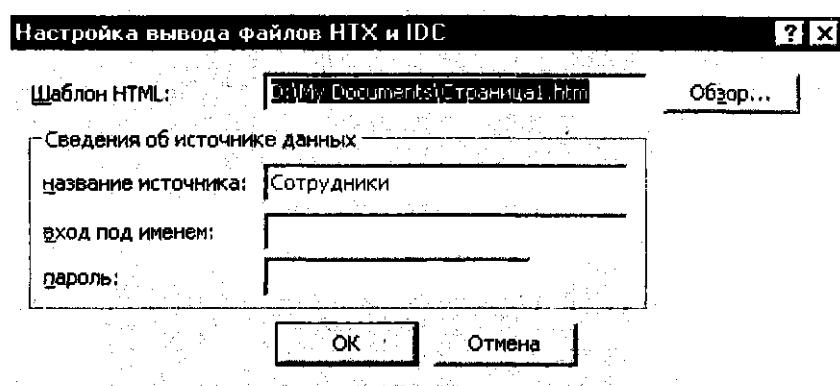


Рис. 18.12. Вид окна Настройка вывода файлов HTX и IDC

- В проекте Microsoft Access в полях **Username** (Имя пользователя) и **Password** (Пароль) ввести имя пользователя и пароль, чтобы разрешить пользователям подключаться к базе данных Microsoft SQL Server с Web-страницы. По умолчанию будут действовать имя пользователя **Sa** и пустой пароль.

Чтобы экспортированная в формат ASP или IDC/HTX таблица (запрос или форма) была доступна пользователям Интернета, необходимо опубликовать ее. Для этого нужно выполнить следующее:

- Установить программное обеспечение (Microsoft Internet Information Server или Personal Web Server) на компьютер, используемый в качестве Web-сервера и выполняющий обработку генерируемых сервером файлов HTML.
- Создать в корневом каталоге папку для хранения файлов ASP или IDC/HTX (**\Webshare\Wwwroot** для Personal Web Server и **\Inetpub\Wwwroot** для Microsoft Internet Information Server).
- Установить привилегии доступа к этой папке: **Выполнение** для Windows NT 4.0 и выше; **Выполнение сценариев** для Windows 9x.
- Скопировать в созданную папку файлы ASP или IDC/HTX и все сопутствующие файлы (рисунки, связанные файлы и папки, в которых содержатся эти файлы).
- Скопировать базу данных Microsoft Access в созданную папку или указать ее расположение в сети при задании источника данных ODBC.
- Задать используемый источник данных ODBC в качестве системного источника данных на Web-сервере. Имя этого источника данных совпадает с введенным в соответствующее диалоговое окно Параметры вывода при выводе файлов ASP или IDC/HTX.
- При необходимости определить защиту базы данных для источника данных ODBC, сохранив возможность доступа пользователей к этому источнику. Имя пользователя и пароль должны совпадать со значениями, вводимыми в поля **вход под именем (Username)** и **пароль (Password)** диалоговых окон Параметры вывода файлов ASP (Options Output Files ASP) или Параметры вывода файлов HTX/IDC (Options Output Files HTX/IDC), которые отображаются при выводе файлов ASP или IDC/HTX.

При экспорте таблицы в формат ASP или IDC/HTX данные, видимые при отображении конечного файла .ASP или .HTX в Web-обозревателе, являются фактическими данными набора записей, на котором основана таблица. По умолчанию Microsoft Access сохраняет текстовые поля и поля Memo с выравниванием по левому краю, а все остальные поддерживающие типы данных с выравниванием по правому краю.

При выводе таблицы Microsoft Access не выводит объекты OLE или связанные значения из поля подстановок, а также игнорирует настройки свойств Порядок сортировки (OrderBy), Фильтр (Filter), Формат (Format) и Маска ввода (InputMask).

Характеристика IDC-технологии

Технология IDC свое название получила по имени приложения Internet Database Connector (IDC), входящего в состав Internet Information Server и обеспечивающего связь с драйвером ODBC. Это приложение оформлено в

виде библиотеки динамических связей с именем HTTPODBC.DLL. По существу IDC представляет собой приложение интерфейса Internet Server API (ISAPI), которое упрощает создание динамических (активных) Web-страниц. Напомним, что такие страницы автоматически доставляют обновленную информацию из баз данных, поддерживающих ODBC.

Кроме приложения Internet Database Connector, которое мы не рассматриваем, технологию IDC поддерживает СУБД Access, которая позволяет создавать такие файлы путем экспорта. В этой технологии используются два типа файлов: приложений IDC с расширением .IDC имен и расширенные файлы HTML с расширением имен .HTX, называемые файлами шаблонов.

Файлы IDC определяют источники данных ODBC и запрос, обеспечивающий данные для динамической Web-страницы в обычном текстовом формате. В этих файлах содержатся команды, понимаемые всеми БД, располагающими драйвером ODBC. Кроме того, в этом файле может содержаться необязательная информация о пользователе, к примеру имя, пароль доступа к источнику и т. п.

Файлы шаблонов (.HTX) обеспечивают источники форматирования данных для просмотра. В них могут находиться формы HTML с объектами элементов управления Windows, такими как текстовые поля, поля со списками и т. п. Это позволяет обеспечивать настройку содержимого данных на создаваемых Web-страницах.

При создании динамической Web-страницы по описываемой технологии выполняется объединение данных, обеспечиваемых файлом .IDC, с файлом .HTX. Обычно для каждой создаваемой Web-страницы это объединение выполняется попарно. В принципе данные из одного файла .IDC можно использовать с несколькими файлами шаблонов. При этом драйвер ODBC получает информацию из базы данных, форматирует ее с помощью файла .HTX и формирует Web-страницу. Затем результат возвращается Web-серверу, который отправляет документ клиенту.

Например, пусть требуется создать форму для получения информации от посетителей сайта и сохранения ее в базе данных для последующего использования. Форма должна быть связана с файлом .IDC, как это показано на листинге ниже.

```
<PORM ACTION="/orders/order.idc" METHOD="POST">
Name: <INPUT TYPE="text" NAME="name"XBR>
Street: <INPUT TYPE="text" NAME="street"XBR>
City: <INPUT TYPE="text" NAME="city"XBR>
State: <INPUT TYPE="text" NAME="state">
Zip: <INPUT TYPE="text" NAME="zip" SIZE=10xP>
Select which items you would like to order:<BR>
<INPUT TYPE="checkbox" NAME="ram" VALUE="ram"> 8MB RAM<BR>
<INPUT TYPE="checkbox" NAME="hd" VALUE="hd"> 1.2GB HD<BR>
```

```
<INPUT TYPE="checkbox" NAME="modem" VALUE="modem" > 28.8  
Modem<P>  
<INPUT TYPE="submit" VALUE="Place Order">
```

При нажатии кнопки Place Order (Разместить заказ) введенная в форму информация передается на сервер, который откроет файл ORDER.IDC. Этот файл содержит команды, определяющие базу данных, шаблон .HTX и запрос SQL. Ниже приведено содержимое файла ORDER.IDC для нашего примера:

Datasource:

Orders Template: Orderthx.htm

SQLStatement:INSERT name,street,city,state,zip,ram,hd,modem

Файл .IDC содержит три *обязательные* директивы. Директива **Datasource** указывает базу данных, с которой происходит соединение. Директива **Template** определяет файл .HTX, который должен использоваться для создания HTML-страницы, возвращаемой серверу и в итоге клиенту. Директива **SQLStatement** содержит запрос, отправляющий введенную пользователем информацию в базу данных или получающий оттуда информацию.

Кроме обязательных, файл .IDC может содержать *дополнительные* директивы, повышающие гибкость работы с HTTPODBC.DLL. Ниже описаны дополнительные директивы IDC.

Директива **DefaultParameters** используется для определения параметров по умолчанию, которые используются в случае, если пользователь не заполнил форму целиком. Например, для случая, когда пользователь по какой-то причине не ввел имя для поиска, можно предусмотреть следующую строку:

DefaultParameters: name=%John Doe% В такой строке можно указать несколько параметров, разделенных запятой.

Директива **RequiredParameters** позволяет указать обязательные для заполнения поля формы. Например, если требуется, чтобы обязательно были введены имя и адрес, нужно указать это следующим образом:

RequiredParameters: name, street, city, state, zip

Директива **MaxFieldSize** используется для указания максимальной длины записи. По умолчанию устанавливается длина, равная 8192 байта.

Директива **MaxRecords** позволяет определить максимальное число записей, возвращаемых в ответ на запрос. По умолчанию возвращаются все записи, удовлетворяющие условиям запроса. Однако это можно использовать только в небольших базах данных. Если база данных имеет значительные размеры, следует ограничить этот параметр разумной величиной с учетом типа информации.

Директива **Expires** определяет время (в секундах) до очистки кэш-памяти. Если ее не использовать, то обращение к базе данных происходит при каждом запросе информации. При использовании этой директивы ответ на за-

прос возвращается из кэш-памяти. Это позволяет уменьшить нагрузку системы и увеличить скорость получения информации.

Директива **Username** позволяет указать имя для доступа к SQL-серверу, если не используется встроенная система безопасности SQL-сервера.

Директива **Password** позволяет установить парольную защиту. При указании этой директивы требуется ввести имя пользователя.

Рассмотрим особенности построения файла с расширением .HTX, который создает HTML-документ, возвращаемый клиенту. Как и IDC, файл HTX имеет специальные команды или теги, которые помогают отформатировать страницу.

При обращении к базе данных сохранение возвращаемой информации выполняется с помощью тегов `<%begindetail%>` и `<%enddetail%>`. Предположим, что при просмотре каталога продукции вводится запрос для поиска информации о модемах, которая содержится в поле под именем `modem`. В этом случае можно создать следующий файл .HTX для форматирования результата поиска:

```
<table>
<%begindetail%>
<tr><td><%modem%><td><%price%></td></tr>
<%enddetail%>
</table>
```

Приведенный код открывается тегом `<TABLE>`. При совпадении запроса и записи создается строка, состоящая из названия модема и его цены. Тег `<%enddetail%>` указывает на окончание этого раздела кода. Затем тег `</TABLE>` закрывает таблицу. Если ни одной записи не найдено, этот код пропускается.

Директива **CurrentRecord** подсчитывает количество обработок записей. Она может использоваться для проверки, сгенерированы ли в ответ на запрос какие-либо результаты, и для информирования пользователя об этом.

Внутри файла .HTX можно употреблять следующие простые условные операторы: `<%if%>`, `<%else%>` и `<%endif%>`. Они позволяют проверить выполнение каких-либо условий. Например, можно проверить, возвращены ли результаты поиска, если нет, проинформировать об этом пользователя:

```
<table>
<%begindetail%>
<tr><td><%modem%><td><%price%></td></tr>
<%enddetail%>
</table>
<%if CurrentRecord EQ 0 %>
There isn't anything in the database that matches your query.
<ce,rlter>
```

```
<a href="products.html">[Product Database]</a>
</center>
```

Тег `<%if%>` для проверки информации может использовать четыре условия. С помощью условия `EQ` проверяется равенство:

```
<%if modem EQ "US Robotics" %>
US Robotics 28.8
<%endif%>
```

С помощью условия `GT` проверяется, больше ли одно значение другого:

```
<%if price GT 500 %>
```

С помощью условия `LT` проверяется, меньше ли одно значение другого:

```
<%if price LT 10 %>
```

Условие `CONTAINS` проверяет, содержится ли одно значение в другом:

```
<%if modem CONTAINS "Robotics" %>
US Robotics
<%endif%>
```

Переменная `MaxRecords` содержит значение директивы `MaxRecords`:

```
<%if CurrentRecord EQ MaxRecords %> Results have been abridged
<%endif%>
```

После заполнения пользователем формы HTML можно направить ее непосредственно в файл .HTX посредством префикса `idc`. Например, для возврата пользователю введенной им информации, нужно указать такой код:

```
Hello %idc.name%. How is the weather in %idc.city%,
%idc.state%?<BR>
```

В файле .HTX можно использовать переменные HTTP, которые размещают между символами `<%` и `%>`:

```
<% CurrentRecord%>
```

Для работы с файлами .HTX удобно использовать программу Frontpage из состава пакета Microsoft Office.

Пример публикации по IDC-технологии

Для публикации БД воспользуемся Web-приложением, в состав которого входит главная HTML-страница (с которой начинается выполнение приложения), несколько файлов шаблонов с расширением HTX и файлов запросов к БД с расширением IDC. Для запуска этого приложения использовался Microsoft Personal Web Server, работающий под управлением Windows 98. Файлы приложения размещены в каталоге `c:\WebShare\Scripts`, которому

соответствует виртуальный каталог с псевдонимом Scripts. Приложение рассчитано на работу в обозревателе, который поддерживает JavaScript (например, Microsoft Internet Explore 5.0).

Поясним содержимое и назначение каждого из названных файлов. Главная страница приложения – HTML-страница с именем "my.html" используется для начала работы с приложением, организации взаимодействия между файлами приложения и ввода параметров запроса. Указанная HTML-страница содержит следующий текст:

```
<HTML>
<HEAD>
<TITLE>Пример использования технологии IDC</TITLE>
</HEAD>
<BODY>
<B2><CENTER>Пример использования технологии IDC</
CENTER></B><P>

<FORM name=form1 ACTION="http://localhost/Scripts/form.idc"
METHOD="POST">
Для добавления новой строки в БД необходимо заполнить следующие
поля<BR>
и нажать кнопку добавить:<BR>
Название : <INPUT TYPE="text" NAME="book_title"><BR>
Дата выпуска: <INPUT TYPE="text" NAME="book_data"><BR>
Количество страниц: <INPUT TYPE="text" NAME="book_num"><BR>
Цена: <INPUT TYPE="text" NAME="book_price"><BR>
<INPUT TYPE="Submit" Value="Добавить строку в БД"> <BR><p>
Для удаления строки из БД необходимо ввести название книги и год
издания : <BR>
<INPUT TYPE="text" NAME="del">: <INPUT TYPE="text"
NAME="del1"><BR>
<input type=button value="Удалить строку из БД" NAME="b_del"
onClick="location='http://localhost/Scripts/
books_del.idc?Book='+form1.del.value+'&ProductTime=
'+form1.del1.value"><BR><p>
Для выполнения выборки строки необходимо ввести год издания: <BR>
<INPUT TYPE="text" NAME="year"><BR>
<input type=button value="Выполнение поиска строки в БД"
onClick="location='http://localhost/Scripts/books_1.idc?ProductTime=
'+form1.year.value"><BR><p>
<A href="http://localhost/Scripts/All_types_1.idc">"Просмотр содержимого
всей БД"</A>
```

```
</FORM>
</BODY>
</HTML>
```

Вид рассматриваемой HTML-страницы в окне обозревателя приведен на Рис. 18.13.

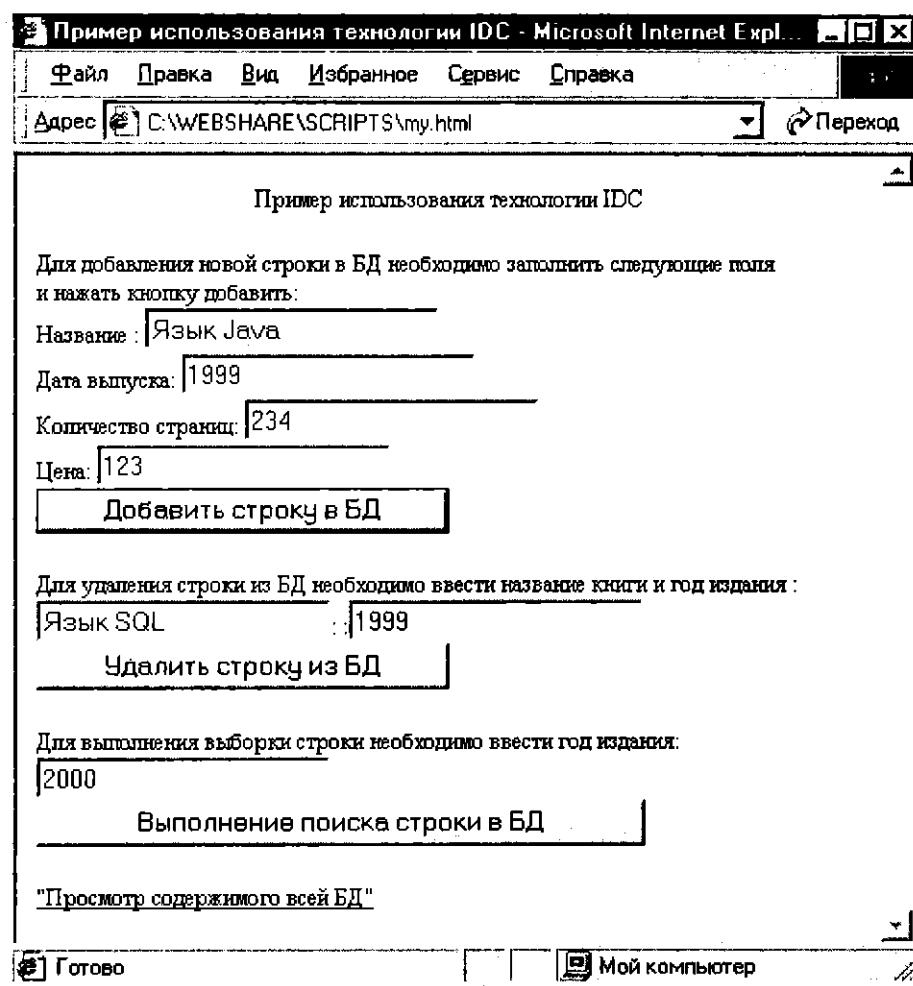


Рис. 18.13. Вид главной HTML-страницы в окне обозревателя

Главная HTML-страница содержит одну форму, включающую ряд интерфейсных элементов, которые по назначению можно разделить на следующие четыре группы.

Первая группа содержит четыре поля с именами book_title, book_data, book_num, book_price, предназначенные для ввода параметров запроса, с помощью которого **добавляется** строка в БД.

В эту же группу входит кнопка, имеющая тип “Submit” и значение “Добавить строку в БД”. Эта кнопка служит для запуска обработчика запроса обозревателя к серверу. В качестве обработчика здесь выступает файл, загружаемый по следующему URL :

“<http://localhost/Scripts/form.idc>”

Последний записан в атрибуте тега формы:

```
<FORM name=form1 ACTION="http://localhost/Scripts/form.idc"  
METHOD="POST">
```

Названный обработчик представляет собой IDC-файл, который содержит параметры соединения с источником, SQL-запрос на добавление строки к БД, ссылку на используемый HTX-шаблон и другие параметры. Файл “form.idc” содержит следующие строки текста:

```
Datasource:books_idc  
Template:form.htm  
RequiredParameters:book_title,book_data,book_num,book_price  
SQLStatement:INSERT INTO Books (book,ProductTime,Price,StrNum)  
values  
('%book_title%', '%book_data%', '%book_num%', '%book_price%')
```

Строка Datasource:books_idc здесь (и в других IDC-файлах) указывает серверу псевдоним источника данных – “books_idc”, зарегистрированный в администраторе ODBC, связанный с базой данных в формате MS Access. В общем случае эта база данных может иметь формат любой СУБД с драйвером ODBC.

Строка SQLStatement задает SQL-запрос на добавление записи в БД. В ней подстрока

```
('%book_title%', '%book_data%', '%book_num%', '%book_price%')
```

содержит специальные команды сервера, заключенные в символы “%”. Внутри них помещаются имена параметров, передаваемых через HTTP-запрос. Вместо этих команд сервер подставляет значения параметров, переданных обозревателем серверу. Далее этот запрос передается функциям ODBC для выполнения. Стока RequiredParameters:book_title,book_data,book_num,book_price указывает обязательные параметры, которые должны передаваться обозревателем серверу в запросе, и статические данные, которые непосредственно выводятся в выходной поток сервера.

Строка Template:form.htm указывает серверу, что в качестве шаблона будет использоваться файл form.htm. Этот файл используется для генерации возвращаемого файла в качестве ответа обозревателю на запрос. В нашем случае в результате выполнения запроса данные не возвращаются. При этом в каче-

стве ответа обозревателю, для демонстрации возможностей технологии IDC/HTX, генерируется HTML-документ с информацией о переменных сервера.

Файл form.htm содержит следующий текст:

```
<HTML>
<HEAD></HEAD>
<BODY>
Запрос выполнен!<BR>
Значения системных переменных:<BR>
Метод передачи данных: <%REQUEST_METHOD%><BR>
<%if REQUEST_METHOD CONTAINS "POST" %>
Длина данных, передаваемых через поток:
<%CONTENT_LENGTH%><BR>
Тип данных:
<%CONTENT_TYPE%><BR>
<%else%>
Данные переданные методом <%REQUEST_METHOD%>;
<%QUERY_STRING%><BR>
<%endif%>
Версия программного обеспечения сервера:
<%SERVER_SOFTWARE%><BR>
Виртуальный путь: <%SERVER_NAME%><BR>
<A href="javascript:history.back()">
"Назад"</A>
</BODY>
</HTML>
```

В приведенном файле для вывода значений переменных сервера используются специальные теги <%Имя переменной%>. Например, для вывода значения переменной REQUEST_METHOD используется строка <%REQUEST_METHOD%>.

Конструкция

```
<%if REQUEST_METHOD CONTAINS "POST" %>
...
<%else%>
...
<%endif%>
```

служит для вывода значений различных переменных сервера в зависимости от используемого метода передачи данных.

В конце файла form.htm находится ссылка с помощью, которой выполняется возврат на главную страницу Web-приложения:

```
<A href="javascript:history.back()">
"Назад"</A>
```

Таким образом, первая группа интерфейсных элементов вместе с IDC/HTX-файлами обеспечивает выполнение запроса на *добавление* записей в БД. При этом для выполнения запроса нужно заполнить все четыре поля и нажать кнопку «Добавить строку в БД». При этом сервер обрабатывает запрос обозревателя и в случае успешного выполнения возвращает HTML-страницу, вид которой в окне обозревателя показан на рис. 18.14.

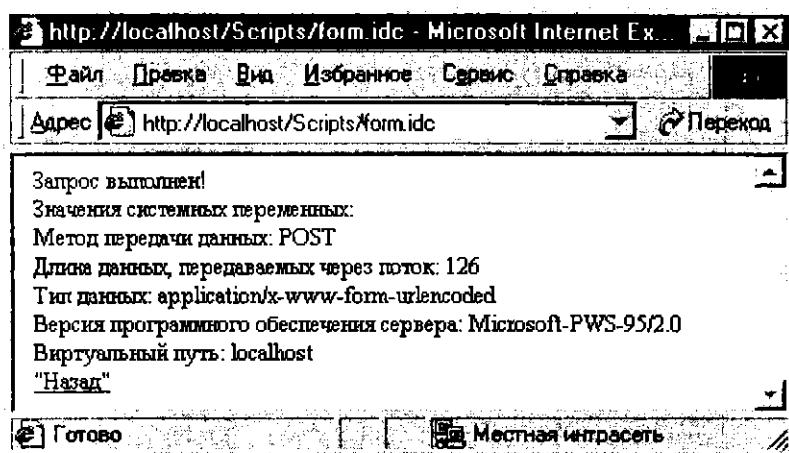


Рис. 18.14 Окно обозревателя с результатом обработки файла form.idc

В указанном окне (рис. 18.14) сначала выводятся значения переменных, заполняемых сервером при использовании метода передачи данных POST. Это переменные **CONTENT_LENGTH** и **CONTENT_TYPE**. Кроме того, независимо от метода передачи данных выводятся значения переменных **REQUEST_METHOD**, **SERVER_SOFTWARE** и **SERVER_NAME**.

Вторая группа интерфейсных элементов главной страницы приложения содержит два поля с именами **del**, **del1**, предназначенные для ввода параметров запроса, с помощью которого удаляется строка из БД. Кроме того, к этой группе относится кнопка с именем **b_del** и значением «Удалить строку из БД». Второй группе интерфейсных элементов приложения соответствуют следующие строки текста главной страницы приложения:

```
<INPUT TYPE="text" NAME="del"> : <INPUT TYPE="text"
NAME="del1"><BR>
<input type=button value="Удалить строку из БД" NAME="b_del"
onClick="location='http://localhost/Scripts/
books_del.idc?Book=' +form1.del.value + '&ProductTime=' +form1.del1.value"><BR><p>
```

Кнопка с именем **b_del** служит для запуска обработчика запроса обозревателя к серверу. В нашем случае в качестве обработчика выступает файл, загружаемый по следующему URL:

`"http://localhost/Scripts/books_del.idc"`

Для загрузки этой страницы используется следующий код на JavaScript:

```
onClick="location='http://localhost/Scripts/  
books_del.idc?Book='+form1.del.value+'&ProductTime='+form1.del1.value"
```

Этот код выполняется при обработке события **onClick**, возникающего при нажатии на кнопку с помощью мыши. Здесь объекту **location** присваивается новый URL, что приводит к загрузке сервером требуемого обработчика.

При этом к URL “`http://localhost/Scripts/books_del.idc`” с помощью конкатенации строк добавляется подстрока с передаваемыми параметрами запроса с использованием метода **GET**

`"+form1.del.value+'&ProductTime='+form1.del1.value"`

Здесь код `form1.del.value` и `form1.del1.value` возвращают содержимое поля **del** и **del1** соответственно. Параметры **Book** и **ProductTime** совпадают с названиями столбцов БД и содержат название книги и год издания соответственно.

Обработчик “`books_del.idc`” является IDC-файлом, который содержит параметры соединения с источником, SQL-запрос для удаления строк из БД, ссылку на используемый HTX-шаблон и другие параметры. Файл “`books_del.idc`” содержит следующий текст:

Содержимое файла “`Books_del.idc`”:

`Datasource:Books_idc`

`Template:form.htm`

`RequiredParameters:Book,ProductTime`

`SQLStatement: DELETE * FROM [Books] WHERE Book='%Book%' AND
ProductTime=%ProductTime%`

В приведенном тексте строка `SQLStatement` задает SQL-запрос на удаление записи из БД:

`DELETE * FROM [Books] WHERE Book='%Book%' AND
ProductTime=%ProductTime%`

В этой строке подстрока

`Book='%Book%' AND ProductTime=%ProductTime%`

определяет выбор строки из БД по совпадению значений полей (подстроки “`Book=`” и “`ProductTime=`”) в базе данных и значений передаваемых па-

метров запроса Book и ProductTime, подставляемых вместо специальных IDC-команд %Book% и %ProductTime%.

Строка

RequiredParameters:Book,ProductTime

указывает обязательные параметры, которые должны передаваться обозревателем серверу в запросе.

Строка Template:form.htm указывает серверу, что в качестве шаблона будет использоваться файл form.htm. Этот файл используется для генерации файла, возвращаемого в качестве ответа обозревателю на запрос, аналогично предыдущей группе интерфейсных элементов. Единственное отличие заключается в том, что здесь при вызове сработчика для передачи данных используется метод GET (рис. 18.15).

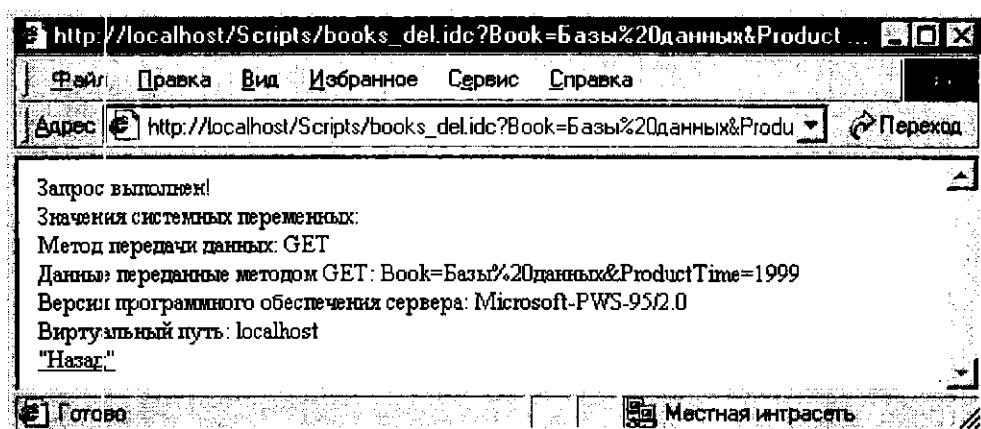


Рис. 18.15. Окно обозревателя с результатом обработки файла books_del.idc

В соответствии с изложенным в результирующий HTML-документ выводятся значения переменных сервера (REQUEST_METHOD и QUERY_STRING), заполняемые при использовании метода GET:

Данные, переданные методом <%REQUEST_METHOD%>:

<%QUERY_STRING%>

Таким образом, вторая группа интерфейсных элементов вместе с IDC/HTX-файлами обеспечивает выполнение запроса на удаление записей из БД. При этом для выполнения запроса нужно заполнить два текстовых поля и нажать кнопку «Удалить строку в БД».

Третья группа интерфейсных элементов главной страницы приложения содержит одно поле с именем ueag, предназначенное для ввода параметра запроса, с

помощью которого находится строка в БД. Поиск строки в БД осуществляется по полю, содержащему год издания. Кроме того, к этой группе относится кнопка с именем **b_year**, имеющая значение «Выполнение поиска строки в БД».

Третьей группе интерфейсных элементов приложения соответствуют следующие строки текста главной страницы приложения:

```
<INPUT TYPE="text" NAME="year"><BR>
<input type=button value="Выполнение поиска строки в БД"
NAME="b_year"
onClick="location='http://localhost/Scripts/
books_1.idc?ProductTime='+form1.year.value"><BR><p>
```

Кнопка с именем **b_year** служит для запуска обработчика запроса обозревателя к серверу. В этом случае в качестве обработчика выступает файл, загружаемый по следующему URL:

[“http://localhost/Scripts/books_1.idc”](http://localhost/Scripts/books_1.idc)

Для загрузки этой страницы используется следующий код на JavaScript

```
onClick="location='http://localhost/Scripts/
books_1.idc?ProductTime='+form1.year.value"
```

Он выполняется при обработке события **onClick**, возникающего при нажатии на кнопку с помощью мыши. В этом коде объекту **location** присваивается новый URL, что приводит к загрузке сервером требуемого обработчика “**books_1.idc**”.

При этом к URL “http://localhost/Scripts/books_1.idc” с помощью конкатенации строк добавляется подстрока с передаваем параметром запроса с использованием метода **GET**:

ProductTime='+form1.year.value

в которой код **form1.year.value** возвращает содержимое поля **year**.

Содержимое файла “**books_1.idc**”:

```
Datasource:Books_idc
Template:Books_1.htm
RequiredParameters:ProductTime
SQLStatement:SELECT * FROM[Books] where
ProductTime=%ProductTime%
```

При этом строка **SQLStatement** задает SQL-запрос на выборку записи из БД:

SELECT * FROM[Books] where ProductTime=%ProductTime%

Здесь подстрока

ProductTime=%ProductTime%

определяет требуемую для выборки БД строку.

Строка

Template:Books_1.htm

указывает серверу, что в качестве шаблона будет использоваться файл Books_1.htm. Этот файл используется сервером для формирования в виде таблицы данных, получаемых в результате выполнения запроса:

Файл Books_1.htm содержит следующий текст:

```
<HTML>
<HEAD>
<TITLE>All_Types</TITLE>
</HEAD>
<BODY>
<TABLE BORDER=1> <CAPTION><B>Books</B></CAPTION>
<THEAD><TR>
<TH>Num_Books></TH>
<TH>Book</TH>
<TH>ProductTime</TH>
<TH>Price</TH>
<TH>StrNum</TH>
</TR></THEAD>
<TBODY>
<%BeginDetail%>
<TR VALIGN=TOP>
<TD><%KodBooks%><BR></TD>
<TD><%Book%><BR></TD>
<TD ALIGN=RIGHT><%ProductTime%><BR></TD>
<TD ALIGN=RIGHT><%Price%><BR></TD>
<TD ALIGN=RIGHT><%StrNum%><BR></TD>
</TR>
<%EndDetail%>
<BR>
<A href="javascript:history.back()">"Назад"</A>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

Основной особенностью этого файла является наличие в нем следующего кода, задающего тело таблицы:

```
<TBODY>
<%BeginDetail%>
```

```
<TR VALIGN=TOP>
<TD><%КодBooks%><BR></TD>
<TD><%Book%><BR></TD>
<TD ALIGN=RIGHT><%ProductTime%><BR></TD>
<TD ALIGN=RIGHT><%Price%><BR></TD>
<TD ALIGN=RIGHT><%StrNum%><BR></TD>
</TR>
<%EndDetail%>
```

Использование специальных тегов <%BeginDetail%>...<%EndDetail%> задает для сервера цикл на заполнение строк таблицы. При этом переход от текущей строки таблицы к следующей выполняется автоматически. Таким образом, в результате обработки этого кода сервер многократно формирует строки таблицы в зависимости от количества строк в результирующем наборе данных.

Специальные теги <%КодBooks%>, <%Book%>, <%ProductTime%>, <%Price%> и <%StrNum%> используются для указания серверу мест вывода значений соответствующих полей из записей БД.

В конце файла Books_1.htm находится ссылка

```
<A href="javascript:history.back()">"Назад"</A>
```

С ее помощью выполняется возврат на главную страницу Web-приложения.

Таким образом, третья группа интерфейсных элементов вместе с IDC/HTX-файлами обеспечивает выполнение запроса на выборку записей из БД. При этом для успешного выполнения запроса нужно заполнить текстовое поле, задающее год издания книги, и нажать кнопку «Выполнение поиска строки в БД».

Например, при задании в текстовом поле year значения 2000 (рис. 18.13) и нажатии на кнопку b_year в окне обозревателя будет выведен текст, представленный на рис. 18.16.

На этом рисунке выведена одна строка, так как приведенному в запросе параметру в БД соответствует только одна строка, удовлетворяющая условию поиска.

Четвертая группа интерфейсных элементов главной страницы приложения содержит следующую ссылку:

```
<A href="http://localhost/Scripts/All_types_1.idc">"Просмотр содержимого всей БД"</A>
```

Эта ссылка передает серверу запрос на обработку IDC-файла All_types_1.idc и ссылка служит для получения всех записей из БД. Файл "All_types_1.idc" содержит следующий код:

```
Datasource:Books_idc
Template:Books_1.htm
SQLStatement:SELECT * FROM [Books]
Следующая строка в этом файле
SQLStatement:SELECT * FROM [Books]
```

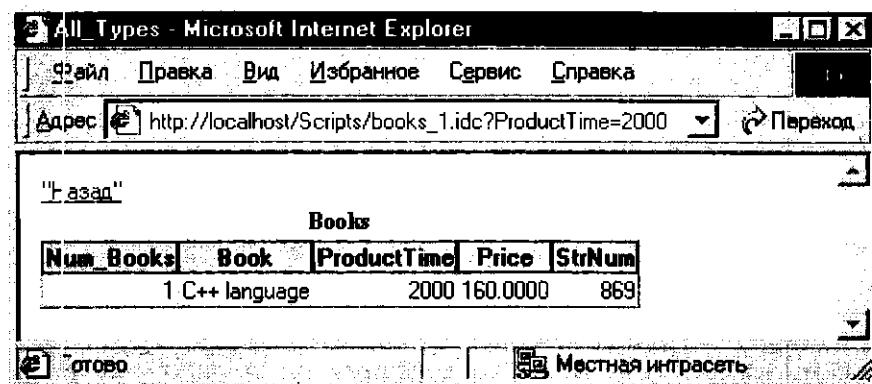


Рис. 18.16. Окно обозревателя с результатом обработки файла books_1.idc

Приведенный код определяет запрос на выборку всех записей из БД. Стока **Template:Books_1.htm**

указывает серверу, что для формирования результирующего HTML-документа будет использоваться тот же файл шаблона, что и в предыдущем случае.

На рис. 18.17 представлены результаты обработки запроса сервером на получение всех строк БД, выводимых в виде таблицы.

Таким образом, мы рассмотрели пример работы по технологии IDC, обеспечив при этом операции добавления, удаления и поиска строк, а также просмотр всего содержимого базы данных.

18.4. Статические файлы HTML

Публикация содержимого БД с использованием статических HTML-страниц является самым простым способом распространения информации из БД в Интернете. При этом для отображения содержимого таблиц, запросов или форм генерируется статическая HTML-страница. Эта страница, как правило, не включает интерфейсные элементы, а основным ее содержимым является статический образ информации из базы данных, представленной в таблице, запросе или форме.

Как отмечалось, статические файлы HTML создаются путем экспорта соответствующих объектов баз данных (таблиц, запросов, форм и отчетов), из которых они создаются.

Публикацию файлов статических HTML-страниц можно выполнить в папках Web или на Web-сервере. Для публикации статических HTML-страниц в папках Web достаточно выполнить копирование в них соответствующих файлов с помощью Проводника Windows.

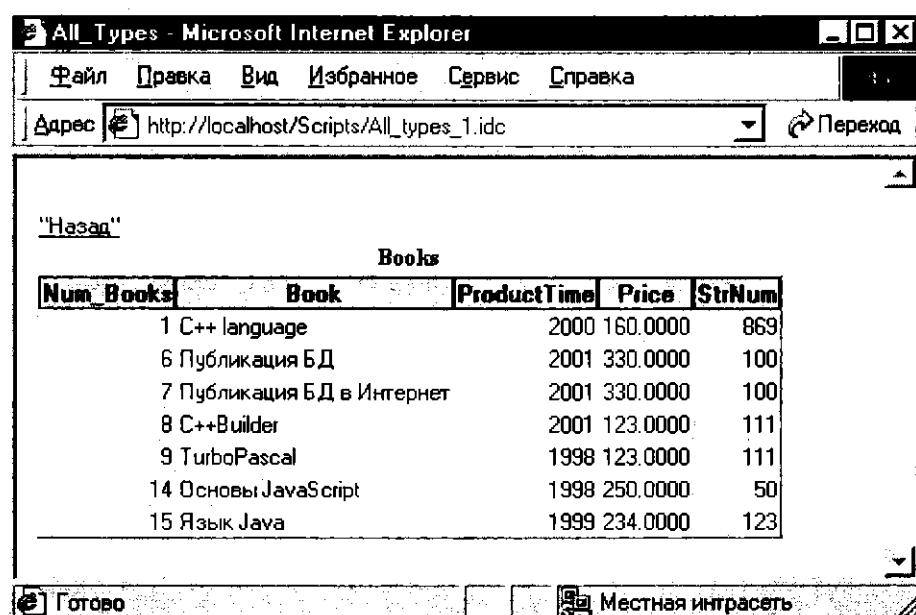


Рис. 18.17 Окно обозревателя с результатом обработки файла All_types.idc

Для публикации статических HTML-страниц на Web-сервере требуется скопировать соответствующие файлы HTML и сопутствующие файлы (рисунки и стили) и папки в папку корневого каталога этого сервера с помощью Проводника. Напомним, что стандартными корневыми каталогами являются \Webshare\Wwwroot для Personal Web Server и \Inetpub\Wwwroot для Microsoft Internet Information Server.

Пример 1. Статическая публикация данных из отчета.

Рассмотрим статическую публикацию данных из отчета **Список товаров**, содержащегося в базе данных Борей СУБД Access 2000. Для решения поставленной задачи выполним следующие действия:

- В окне базы данных выберем имя отчета **Список товаров**, который нужно экспортовать, и в меню **Файл (File)** зададим команду **Экспорт (Export)**.
- В открывшемся окне в поле **Тип файла (File Type)** выберем требуемый вариант **Документы HTML (*.html, *.htm)**.
- Выберем диск и папку, в которую будет экспортирован отчет в виде статического файла HTML.
- В поле **Имя файла (File Name)** введем имя файла и нажмем кнопку **Сохранить (Save)**.
- В диалоговом окне **Параметры вывода в формате HTML (Output Options in HTML Format)** укажем имя и местоположение шаблона,

с использованием которого будет выполняться просмотр сформированного файла HTML в окне обозревателя.

В результате выполненных действий будут сформированы пять HTML-страниц, содержащие информацию из экспортируемого нами отчета. Вид первой из названных страницы в окне обозревателя Internet Explorer показан на рис. 18.18.

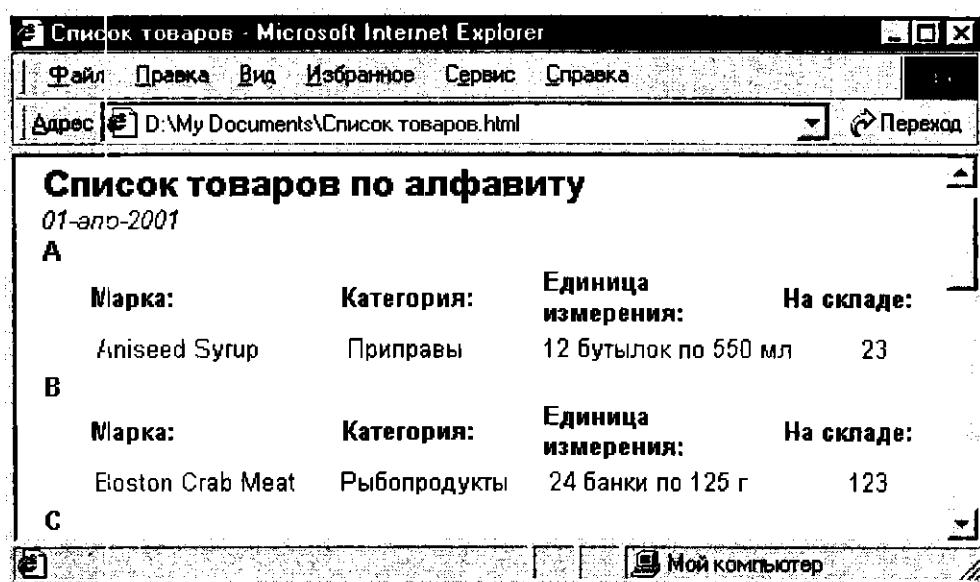


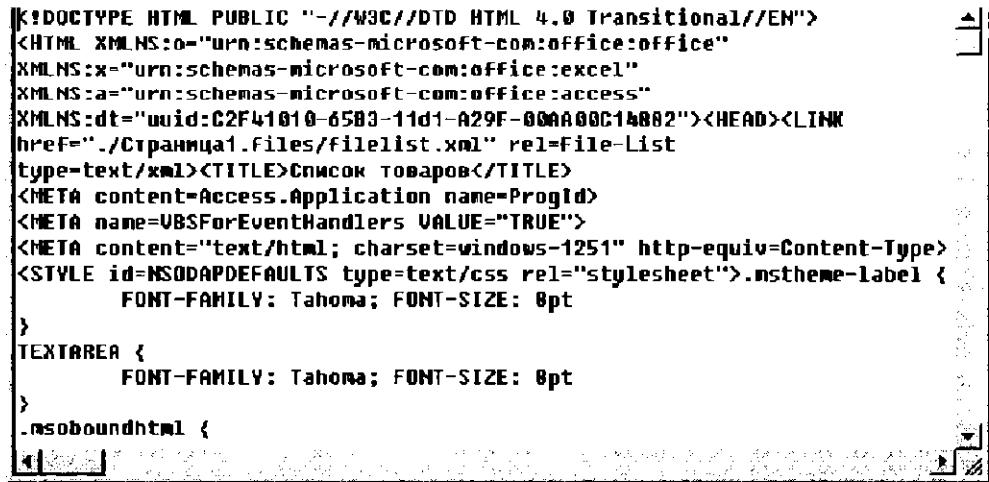
Рис. 18.18. Вид сформированной HTML-страницы в окне обозревателя

Исходный код сформированной нами статической HTML-страницы приведен на рис. 18.19.

Шаблон HTML представляет собой текстовый файл, включающий теги и описатели HTML, являющиеся уникальными для Microsoft Access. Эти описатели указывают места для вставки выходных и других сведений в статические или генерируемые сервером файлы HTML.

При выводе таблицы, запроса, формы или отчета с указанием файла шаблона HTML в связанном диалоговом окне **Параметры вывода (Output Options)** Microsoft Access объединяет шаблон HTML с выходными файлами .HTML, .ASP и .HTX, заменяя описатели как показано в таблице 18.1.

Для публикации файлов статических HTML-страниц в папке Web выполним копирование в нее соответствующих файлов с помощью Проводника Windows.



```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML XMLNS:o="urn:schemas-microsoft-com:office:office"
XMLNS:x="urn:schemas-microsoft-com:office:excel"
XMLNS:a="urn:schemas-microsoft-com:office:access"
XMLNS:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14082"><HEAD><LINK
href=../../../../Страница1.files/filelist.xml" rel=File-List
type=text/xml><TITLE>Список товаров</TITLE>
<META content=Access.Application name=ProgId>
<META name=VBScriptForEventHandlers VALUE="TRUE">
<META content="text/html; charset=windows-1251" http-equiv=Content-Type>
<STYLE id=MSODAPDEFAULTS type=text/css rel="stylesheet">.mstheme-label {
    FONT-FAMILY: Tahoma; FONT-SIZE: 8pt
}
.TEXTAREA {
    FONT-FAMILY: Tahoma; FONT-SIZE: 8pt
}
.msoboundhtml {

```

Рис. 18.19. Исходный код статической HTML-страницы отчета

Таблица 18.1.
Замена описателей шаблона

Описатель шаблона HTML	Замена
<!--AccessTemplate_Title-->	Имя таблицы, запроса, формы или отчета, отображаемое в строке заголовка Web-обозревателя
<!--AccessTemplate_Body-->	Выходные данные таблицы, запроса, формы или отчета
<!--AccessTemplate_FirstPage-->	Тег HTML, содержащий связь с первой страницей отчета
<!--AccessTemplate_PreviousPage -->	Тег HTML, содержащий связь со страницей отчета, предшествующей текущей странице
<!--AccessTemplate_NextPage-->	Тег HTML, содержащий связь со страницей отчета, следующей после текущей страницы
<!--AccessTemplate_LastPage-->	Тег HTML, содержащий связь с последней страницей отчета
<!--AccessTemplate_PageNumber	Номер текущей страницы

Контрольные вопросы и задания

1. Охарактеризуйте разновидности Web-страниц, которые можно создавать с помощью Microsoft Access.
2. Для выполнения каких действий могут использоваться страницы доступа к данным, создаваемые с помощью Microsoft Access?
3. Каково назначение сводных списков?
4. Какие типы разделов используются на страницах доступа к данным с группировкой?
5. Охарактеризуйте создание отчетов в интерактивном режиме.
6. Что нужно учитывать при создании страницы для ввода данных?
7. Укажите последовательность действий при создании сводного списка на странице доступа к данным.
8. Охарактеризуйте серверные страницы, создаваемые с помощью Microsoft Access.
9. Какую последовательность действий нужно выполнить, чтобы экспортированная в формат ASP или IDC/HTX таблица (запрос или форма) была доступна пользователям Интернета?
10. Дайте краткую характеристику IDC-технологии.
11. Каково назначение файлов .HTX?
12. Укажите назначение обязательных и дополнительных директив файла .IDC.
13. Поясните назначение строк кода на главной странице HTML в приведенном в главе примере публикации БД.
14. Как выполняется публикация содержимого БД Access с использованием статических HTML-страниц?
15. Поясните, как при выводе таблицы, запроса, формы или отчета с указанием файла шаблона HTML Microsoft Access выполняет замену описателей шаблона.
16. Выполните средствами СУБД Access создание базы данных и ее публикацию по технологии IDC/HTX.

Литература

1. *Бекаревин Ю., Пушкина Н.* Самоучитель Microsoft Access 2000. СПб.: БХВ – Санкт-Петербург, 1999.
2. *Мещеряков Е. В., Хомоненко А. Д.* Публикация баз данных в Интернете. СПб.: БХВ-Петербург, 2001.
3. *Михеева В., Харитонова И.* Microsoft Access 2000. СПб.: БХВ – Санкт-Петербург, 2000.
4. *Михеева В., Харитонова И.* Microsoft Access 2003. Наиболее полное руководство. – СПб.: БХВ-Петербург, 2004.
5. *Фролов А. В., Фролов Г. В.* Базы данных в Интернете: практическое руководство по созданию Web-приложений с базами данных. М.: Издательско-торговый дом «Русская редакция», 2000.

Приложения

Приложение 1

Аксиомы вывода функциональных зависимостей

Пусть отношение R имеет множество атрибутов $M = \{A_1, A_2, A_3, \dots, A_N\}$ с заданным на нем множеством функциональных зависимостей $F = \{F_1, F_2, F_3, \dots, F_K\}$. X, Y, W и Z - подмножества множества M, то есть атомарные или составные атрибуты.

Аксиома 1 (описывает свойство рефлексивности).

Если $X \subseteq M$, $Y \subseteq M$ и $Y \subseteq X$, то функциональная зависимость $X \rightarrow Y$ следует из F. Это правило позволяет строить тривиальные зависимости типа:

$A_i \rightarrow A_i, A_i A_j \rightarrow A_i, \dots, A_i A_j A_k A_l \rightarrow A_j A_k A_l$ и т. д.

Для их получения достаточно, чтобы множества атрибутов левой и правой частей совпадали, или правая часть была подмножеством левой части.

Пример.

Пусть имеется отношение R(A1, A2, A3, A4, A5), в котором $F = \{A_1 \rightarrow A_2, A_1 \rightarrow A_3, A_1 \rightarrow A_4, A_1 \rightarrow A_5\}$. Отношение R имеет следующий вид:

A1	A2	A3	A4	A5
а	б	а	в	з
д	и	л	г	з
е	б	ш	н	ш
в	к	а	с	х

Образуем составные атрибуты $X = \{A_2, A_4, A_5\}$ и $Y = \{A_2, A_5\}$. Теперь отношение R будет выглядеть следующим образом:

X	Y
б, в, з	б, з
и, г, з	и, з
б, н, ш	б, ш
к, с, х	к, х

Анализируя отношения R, а также атрибуты X и Y, видим, что в R существует функциональная зависимость $X \rightarrow Y$, поскольку одной совокупности значений атрибутов $\{A_2, A_4, A_5\}$ соответствует одна совокупность атрибутов $\{A_2, A_5\}$.

Аксиома 2 (описывает свойство пополнения).

Если $X \subseteq M$, $Y \subseteq M$, $Z \subseteq X$ и задана зависимость $X \rightarrow Y$, которая принадлежит F либо получена из F с использованием правил вывода, то $X \cup Z \rightarrow Y \cup Z$.

Используя эту аксиому, можно любые атрибуты из множества M одновременно подставлять в левую и правую части выражения функциональной зависимости, сохраняя при этом саму зависимость.

Пример.

Пусть $X = \{A1\}$, $Y = \{A3\}$, $Z = \{A4, A5\}$, тогда $X \cup Z \rightarrow Y \cup Z$ то есть $\{A1, A4, A5\} \rightarrow \{A3, A4, A5\}$.

Аксиома 3 (описывает свойство транзитивности).

Если $X \subseteq M$, $Y \subseteq M$, $Z \subseteq X$ и заданы зависимости $X \rightarrow Y$ и $Y \rightarrow Z$, которые принадлежат F либо получены из F с использованием правил вывода, то $X \rightarrow Z$.

Доказательство. Предположим, что в $R(X, Y, Z)$ имеются два кортежа t и s , у которых совпадают значения по атрибутам из множества X (первые столбцы), но не совпадают по атрибутам из множества Z (третий столбцы). Однако это невозможно, так как исходя из зависимости $X \rightarrow Y$ должно быть совпадение значений во вторых столбцах (Y), а зависимость $Y \rightarrow Z$ дает совпадение значений в третьем столбце.

Таким образом, если совпадают значения по атрибуту X , то совпадают значения и по атрибуту Z , а это и означает наличие функциональной зависимости $X \rightarrow Z$.

Аксиомы 1–3 позволяют построить из исходного множества F функциональных зависимостей полное множество F^+ . Из них можно вывести еще пять аксиом.

Аксиома 4 (описывает свойство расширения).

Если $X \subseteq M$, $Y \subseteq M$ и задана функциональная зависимость $X \rightarrow Y$, то для любого $Z \subseteq M$ имеет место зависимость $X \cup Z \rightarrow Y$.

Доказательство. Из аксиомы 1 следует, что $X \cup Z \rightarrow X$, т. к. $X \subseteq X \subseteq Z$. Итак, мы имеем $X \cup Z \rightarrow X$ и заданную из условия зависимость $X \rightarrow Y$. Теперь, если воспользоваться аксиомой 3 о транзитивности, получаем $X \cup Z \rightarrow Y$, что и требовалось доказать.

Аксиома 5 (описывает свойство продолжения).

Если $X \subseteq M$, $Y \subseteq M$, $W \subseteq M$, $Z \subseteq M$ и задана функциональная зависимость $X \rightarrow Y$, то для любых $W \subseteq Z$ имеет место зависимость $X \cup Z \rightarrow Y \cup W$.

Доказательство. Так как $X \rightarrow Y$, то по аксиоме 2 получаем $X \cup Z \rightarrow Y \cup Z$. С другой стороны, условие $W \subseteq Z$ по аксиоме 1 означает, что существует зависимость $Z \rightarrow W$. Из последнего вновь по аксиоме 2 выводится зависимость $Z \cup Y \rightarrow W \cup Y$, которую лучше записать как $Y \cup Z \rightarrow Y \cup W$. Теперь по аксиоме 3 из зависимостей $X \cup Z \rightarrow Y \cup Z$ и $Y \cup Z \rightarrow Y \cup W$ непосредственно следует, что $X \cup Z \rightarrow Y \cup W$.

Аксиома 6 (описывает свойство псевдотранзитивности).

Если $X \subseteq M$, $Y \subseteq M$, $W \subseteq M$, $Z \subseteq M$ и заданы функциональные зависимости $X \rightarrow Y$ и $Y \cup W \rightarrow Z$, то имеет место функциональная зависимость $X \cup W \rightarrow Z$.

Доказательство. Так как $X \rightarrow Y$, то по аксиоме 2 (дополняя по W) можно записать: $X \cup W \rightarrow Y \cup W$. Но так как задано, что $Y \cup W \rightarrow Z$, воспользовавшись аксиомой 3, сразу получим зависимость $X \cup W \rightarrow Z$.

Аксиома 7 (описывает свойство аддитивности или объединения).

Если $X \subseteq M$, $Y \subseteq M$, $Z \subseteq M$ и заданы функциональные зависимости $X \rightarrow Y$ и $X \rightarrow Z$, то имеет место функциональная зависимость $X \rightarrow Y \cup Z$.

Доказательство. Так как $X \rightarrow Y$, то по аксиоме 2 (дополняя по X) получаем: $X \cup X \rightarrow Y \cup X$, или $X \rightarrow Y \cup X$. Из заданной зависимости $X \rightarrow Z$ аналогичным образом (дополняя по Y) получим $X \cup Y \rightarrow Z \cup Y$, которое лучше переписать так: $Y \cup X \rightarrow Y \cup Z$. Из полученных зависимостей $X \rightarrow Y \cup X$ и $Y \cup X \rightarrow Y \cup Z$ по аксиоме 3 следует искомая зависимость $X \rightarrow Y \cup Z$.

Аксиома 8 (описывает свойство декомпозиции).

Если $X \subseteq M$, $Y \subseteq M$, $Z \subseteq M$ и при этом $Z \subseteq Y$, а также задана функциональная зависимость $X \rightarrow Y$, то будет иметь место зависимость $X \rightarrow Z$.

Доказательство. Так как $Z \subseteq Y$, то по аксиоме 1 можно получить $Y \rightarrow Z$. Далее из известной зависимости $X \rightarrow Y$ и только что полученной зависимости $Y \rightarrow Z$ по аксиоме 3 получаем $X \rightarrow Z$.

Приложение 2

Краткое описание языка SQL в MS Access

Операторы (инструкции) языка SQL в СУБД MS Access используются при разработке форм, отчетов, а также написании макрокоманд и программ. Программы в Access хранятся в модулях, написанных на языке программирования Access Basic (в ранних версиях MS Access) либо Visual Basic для приложений, который поглощает язык Access Basic.

К числу основных операторов языка SQL, реализованного в Access, относятся следующие: ALTER TABLE, CREATE INDEX, CREATE TABLE, DELETE, DROP, INSERT INTO, SELECT, TRANSFORM и UPDATE.

Приведем описание инструкций, которое составлено по сведениям справочной системы MS Access.

1. Инструкция ALTER TABLE

Назначение: изменение структуры таблицы, созданной с помощью инструкции CREATE TABLE.

Синтаксис:

```
ALTER TABLE <таблица>
  {ADD {COLUMN <поле> <тип>[(<размер>)] [CONSTRAINT <индекс>] |
    CONSTRAINT <составной_индекс>} |
  DROP {COLUMN <поле> | CONSTRAINT <имя_индекса>}}
```

Аргументы:

<таблица> – имя изменяемой таблицы;

<поле> – имя поля, добавляемого в таблицу или удаляемого из нее;

<тип> – тип данных поля;

<размер> – размер поля в символах (для текстовых и двоичных полей);

<индекс> – индекс для поля. Для получения более подробных сведений о создании индекса смотрите описание предложения CONSTRAINT (п. 11 приложения);

<составной_индекс> – описание составного индекса, добавляемого к таблице;

<имя_индекса> – имя составного индекса, который следует удалить.

Пример.

Удалить поле «Оклад» из таблицы «Сотрудники», которая создана ранее с помощью инструкции CREATE TABLE.

```
ALTER TABLE Сотрудники DROP COLUMN Оклад;
```

2. Инструкция CREATE INDEX

Назначение: создание нового индекса для существующей таблицы.

Синтаксис:

```
CREATE [ UNIQUE ] INDEX <индекс>
  ON <таблица> (<поле> [ASC|DESC][, <поле> [ASC|DESC], ...])
  [WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

Аргументы:

<индекс> – имя создаваемого индекса;
<таблица> – имя существующей таблицы, для которой создается индекс;
<поле> – имена одного или нескольких полей, включаемых в индекс. Для создания простого индекса (состоящего из одного поля) введите имя поля в круглых скобках сразу после имени таблицы. Для создания составного индекса (состоящий из нескольких полей) перечислите имена всех этих полей. Для расположения элементов индекса в убывающем порядке используйте зарезервированное слово **DESC**; в противном случае будет принят возрастающий порядок.

Пример.

Создать в таблице «Клиенты» индекс по полю «КодКлиента». В этом поле исключить повторяющиеся и пустые значения.

```
CREATE UNIQUE INDEX ИндексКлиента ON Клиенты (КодКлиента)
WITH DISALLOW NULL;
```

3. Инструкция CREATE TABLE

Назначение: создание новой таблицы.

Синтаксис:

```
CREATE TABLE <таблица> (
    <поле1> <тип> [(<размер>)] [<индекс1>]
    [, <поле2> <тип> [(<размер>)] [<индекс2>] [, ...]]
    [, <составной_индекс>[, ...]])
```

Аргументы:

<таблица> – имя создаваемой таблицы;
<поле1>, <поле2> – имена одного или нескольких полей, создаваемых в новой таблице. Таблица должна содержать хотя бы одно поле;
<тип> – тип данных поля в новой таблице;
<размер> – размер поля в символах (для текстовых и двоичных полей);
<индекс1>, <индекс2> – предложение **CONSTRAINT**, предназначенное для создания простого индекса (см. п. 11 приложения);
<составной_индекс> – предложение **CONSTRAINT**, предназначенное для создания составного индекса (см. п. 11 приложения);

Пример.

Создать новую таблицу «Таблица1» с двумя текстовыми полями и одним целочисленным полем. Поле «Страховка» сделать ключевым.

```
CREATE TABLE Таблица1 (Имя TEXT, Фамилия TEXT, Страховка
    INTEGER CONSTRAINT Индекс1 PRIMARY KEY);
```

4. Инструкция DELETE

Назначение: создание запроса на удаление записей из одной или нескольких таблиц, перечисленных в предложении **FROM**, которые удовлетворяют предложению **WHERE**.

Синтаксис:

```
DELETE [<таблица>.*]
FROM <таблица>
WHERE <условие_отбора>
```

Аргументы:

<таблица> – необязательное имя таблицы, из которой удаляются записи;

<таблица> – имя таблицы, из которой удаляются записи;

<условие_отбора> – выражение, определяющее удаляемые записи.

Пример.

Удалить записи о всех сотрудниках, которые занимают должность «Стажер» и имеют запись в таблице «Оплата». Между таблицами «Сотрудники» и «Оплата» установлена связь 1:1.

```
DELETE Сотрудники.* FROM Сотрудники INNER JOIN Оплата
    ON Сотрудники.КодСотрудника = Оплата. КодСотрудника
    WHERE Сотрудники.Должность = 'Стажер';
```

5. Инструкция DROP

Назначение: удаление таблицы из базы данных или индекса из таблицы.

Синтаксис:

```
DROP {TABLE <таблица> | INDEX <индекс> ON <таблица>}
```

Аргументы:

<таблица> – имя таблицы, которую следует удалить или из которой следует удалить индекс;

<индекс> – имя индекса, удаляемого из таблицы.

Пример.

Удалить «Индекс1» из таблицы «Стажеры».

```
DROP INDEX Индекс1 ON Стажеры;
```

6. Инструкция INSERT INTO

Назначение: добавить запись или записи в таблицу. Эта инструкция обра-зует запрос на добавление.

Синтаксис:

Запрос на добавление нескольких записей:

```
INSERT INTO <назначение>
    [IN <внешняя_база_данных>] [(<поле1>[,<поле2>[, ...]])]
    SELECT [<источник>]<поле1>[,<поле2>[, ...]]
    FROM <выражение>
```

Запрос на добавление одной записи:

```
INSERT INTO <назначение> [(<поле1>[,<поле2>[, ...]])]
VALUES (<значение1>[, <значение2>[, ...]])
```

Аргументы:

<назначение> – имя таблицы или запроса, в которые добавляются записи;

<внешняя_база_данных> – путь к внешней базе данных. Более подробные сведения об этом аргументе находятся в описании предложения **IN**;

<источник> – имя таблицы или запроса, откуда копируются записи;

<поле1>, <поле2> – имена полей для добавления данных, если они следуют за аргументом <назначение>; имена полей, из которых берутся данные, если они следуют за аргументом источник;

<выражение> – имена таблицы или таблиц, откуда вставляются данные. Выражение может быть именем отдельной таблицы или результатом операции **INNER JOIN**, **LEFT JOIN** или **RIGHT JOIN** или сохраненным запросом;

<значение1>, <значение2> – значения, добавляемые в указанные поля новой записи. Каждое значение вставляется в поле, занимающее то же положение в списке: <значение1> будет вставлено в <поле1>, <значение2> – в <поле2> и т. п. Каждое значение заключается в кавычки (" "); для разделения значений используются запятые.

Пример.

Отобрать все записи из таблицы «Стажеры» для стажеров, принятых на работу более 30 дней назад, и добавить их в таблицу «Сотрудники».

```
INSERT INTO Сотрудники SELECT Стажеры.* FROM Стажеры
WHERE ДатаНайма < Now() - 30;
```

7. Инструкция SELECT

Назначение: представить данные из базы данных в виде набора записей.

Синтаксис:

```
SELECT [<предикат>] { * | <таблица>.* | [<таблица.>]<поле1>
[AS <псевдоним1>] [, [<таблица>].<поле2> [AS <псевдоним2>] [, ...]]}
FROM <выражение> [, ...] [IN <внешняя_база_данных>]
[WHERE...]
[GROUP BY...]
[HAVING...]
[ORDER BY...]
[WITH OWNERACCESS OPTION]
```

Аргументы:

<предикат> – один из следующих предикатов отбора: **ALL**, **DISTINCT**, **DISTINCTROW** или **TOP**. Предикаты используются для ограничения числа возвращаемых записей. Если они отсутствуют, по умолчанию используется предикат **ALL**.

* – указывает, что выбраны все поля заданной таблицы или таблиц;

<таблица> – имя таблицы, из которой должны отбирать записи;

<поле1>, <поле2> – имена полей, из которых должны отбирать данные.

Если включить несколько полей, они будут извлекаться в указанном порядке;
 <псевдоним1>, <псевдоним2> – имена, которые станут заголовками столбцов вместо исходных названий столбцов в таблице;
 <выражение> – имена одной или нескольких таблиц, которые содержат отбираемые данные;
 <внешняя_база_данных> – имя базы данных, которая содержит таблицы, указанные с помощью предыдущего аргумента, если они находятся не в текущей базе данных.

Примеры.

Отобрать все поля из таблицы «Сотрудники».

SELECT Сотрудники.* FROM Сотрудники;

Подсчитать число записей, которые содержат непустое значение в поле «Индекс», и присвоить заголовок «Итого» полю, в которое возвращается результат.

SELECT Count(Индекс) AS Итого FROM Клиенты;

Вызвести число сотрудников и их среднюю и максимальную зарплату.

SELECT Count(*) AS ЧислоСотрудников, Avg(Оклад) AS Средний оклад, Max(Оклад) AS Максимальный оклад FROM Сотрудники;

8. Инструкция SELECT...INTO

Назначение: подготовка запроса на создание таблицы. Запрос на создание таблицы можно использовать для архивации записей, создания резервных копий таблицы, копий для экспорта в другую базу данных или основы отчета, отображающего данные за конкретный период времени.

Синтаксис:

```
SELECT <поле1>[,<поле2>[,...]]  
INTO <новая_таблица> [IN <внешняя_база_данных>]  
FROM <источник>
```

Аргументы:

<поле1>, <поле2> – имена полей, которые следует скопировать в новую таблицу;

<новая_таблица> – имя создаваемой таблицы. Должно удовлетворять стандартным правилам именования. Если <новая_таблица> совпадает с именем существующей таблицы, возникает устранимая ошибка;

<внешняя_база_данных> – путь к внешней базе данных. Более подробные сведения об этом аргументе приведены в описании предложения IN;

<источник> – имя существующей таблицы, из которой отбираются записи. Это может быть одна таблица, несколько таблиц или запрос.

Примеры.

Создать таблицу с именем «Стажеры» и скопировать в нее записи о всех сотрудниках, имеющих должность «Стажер».

```
SELECT Сотрудники.Имя, Фамилия INTO Стажеры FROM Сотрудники
WHERE Должность = 'Стажер';
```

Создать таблицу, содержащую сведения о всех стажерах и их зарплате. Между таблицами «Сотрудники» и «Оплата» установлена связь 1:1. Новая таблица должна содержать все данные из таблицы «Сотрудники», а также данные поля «Оклад» из таблицы «Оплата».

```
SELECT Сотрудники.*, Оклад INTO Стажеры FROM Сотрудники
INNER JOIN Оплата ON Сотрудники.КодСотрудника =
Оплата.КодСотрудника WHERE Должность = 'Стажер';
```

9. Инструкция TRANSFORM

Назначение: создание перекрестного запроса (запрос, возвращающий данные в виде электронной таблицы, используя указанные поля как заголовки строк и столбцов, и способный возвращать итоговые данные). Перекрестный запрос позволяет просматривать данные в более компактной форме, чем при работе с запросом на выборку.

Синтаксис:

```
TRANSFORM <стат_функция>
<инструкция>
PIVOT <поле> [IN (<значение1>[,<значение2>[, ...]])]
```

Аргументы:

<стат_функция> – статистическая функция SQL, обрабатывающая указанные данные;

<инструкция> – инструкция SELECT;

<поле> – поле или выражение, которое содержит заголовки столбцов для результирующего набора;

<значение1>, <значение2> – фиксированные значения, используемые при создании заголовков столбцов.

Пример.

Создать перекрестный запрос, показывающий распределение продаж по месяцам указанного пользователем года. Месяцы должны определять заголовки столбцов слева направо, а марка товаров – заголовки строк сверху вниз.

```
PARAMETERS [Год продажи?] LONG;
TRANSFORM
Sum(Заказано.Количество * (Заказано.Цена - (Заказано.Скидка / 100)
* Заказано.Цена)) AS Продажи
SELECT Марка
FROM Заказы INNER JOIN
(Товары INNER JOIN Заказано ON Товары.КодТовара =
Заказано.КодТовара) ON Заказы.КодЗаказа = Заказано.КодЗаказа
```

```
WHERE DatePart("yyyy", ДатаРазмещения) = [Год продажи?]
GROUP BY Марка
ORDER BY Марка
PIVOT DatePart("m", ДатаРазмещения);
```

В этом примере перед инструкцией **TRANSFORM** стоит оператор **PARAMETERS**, который запрашивает у пользователя значение переменной «Год продажи?». Это позволяет построить запрос с параметром.

10. Инструкция UPDATE

Назначение: создание запроса на обновление записей, который изменяет значения полей указанной таблицы на основе заданного условия отбора.

Синтаксис:

```
UPDATE <таблица>
SET <новое_значение>
WHERE <условие_отбора>;
```

Аргументы:

<таблица> – имя таблицы, данные в которой следует изменить;

<новое_значение> – выражение, определяющее значение, которое должно быть вставлено в указанное поле обновленных записей;

<условие_отбора> – выражение, отбирающее записи, которые должны быть изменены. При выполнении этой инструкции будут изменены только записи, удовлетворяющие этому условию.

Пример.

Увеличить на 10 процентов цену на все товары поставщика, имеющего код 8, поставки которых еще не прекращены.

```
UPDATE Товары SET Цена = Цена * 1.1 WHERE КодПоставщика = 8
AND ПоставкиПрекращены = No;
```

11. Предложение CONSTRAINT

Назначение: создание или удаление индексов в инструкциях **ALTER TABLE** и **CREATE TABLE**. С его помощью можно создавать или удалять простой индекс (по одному полю) или составной индекс (по нескольким полям).

Синтаксис предложения **CONSTRAINT** для создания простого индекса:

```
CONSTRAINT <имя> {PRIMARY KEY | UNIQUE |
REFERENCES <внешняя_таблица> [<внешнее_поле 1>,
<внешнее_поле 2>]}>
```

Синтаксис предложения **CONSTRAINT** для создания составного индекса:

```
CONSTRAINT <имя>
{PRIMARY KEY (<ключевое1>[, <ключевое2> [, ...]]) | 
UNIQUE (<的独特性1>[, <独特性2> [, ...]]) |
```

```
FOREIGN KEY (<ссылка1>[, <ссылка2> [, ...]])  
REFERENCES <внешняя_таблица> [<внешнее_поле1>  
[, <внешнее_поле2> [, ...]])]}
```

Аргументы:

<имя> – имя индекса, который следует создать;

<ключевое1>, <ключевое2> – имена одного или нескольких полей, которые следует назначить ключевыми;

<уникальное1>, <уникальное2> – имена одного или нескольких полей, которые следует включить в уникальный индекс;

<ссылка1>, <ссылка2> – имена одного или нескольких полей, включенных во внешний ключ, которые ссылаются на поля в другой таблице;

<внешняя_таблица> – имя внешней таблицы, которая содержит поля, указанные с помощью аргумента <внешнее_поле>;

<внешнее_поле1>, <внешнее_поле2> – имена одного или нескольких полей во внешней таблице, на которые ссылаются поля, указанные с помощью аргумента <ссылка1>, <ссылка2>. Это предложение можно опустить, если данное поле является ключом внешней таблицы.

Замечания:

- не следует использовать зарезервированные слова PRIMARY KEY при создании индекса в таблице, в которой определен ключ;
- нельзя добавить или удалить одновременно несколько полей или индексов;
- инструкцию CREATE INDEX можно использовать для добавления к таблице простого или составного индекса, а инструкции ALTER TABLE и DROP – для удаления индекса, созданного с помощью ALTER TABLE или CREATE INDEX;
- при добавлении индекса указываются все необходимые сведения об индексе, а при его удалении – достаточно указать его имя;
- зарезервированное слово UNIQUE используется для обеспечения уникальности значений в поле;
- зарезервированные слова PRIMARY KEY используются для создания ключа таблицы, состоящего из одного или нескольких полей. Все значения в ключевом поле таблицы должны быть уникальными и не Null. В таблице может быть только один ключ;
- зарезервированные слова FOREIGN KEY используются для создания внешнего ключа.

Перечень терминов

Администратор базы данных (database administrator) – лицо или группа лиц, отвечающих за выработку требований к БД, ее проектирование, создание, эффективное использование и сопровождение.

Анимация (animation) – метод имитации движения, состоящий в последовательном воспроизведении изображений.

Аномалия модификации (anomaly of modification) – проявляется в том, что изменение значения одного данного может повлечь за собой просмотр всей таблицы и соответствующее изменение других записей таблицы.

Аномалия удаления (anomaly of deletion) – проявляется в том, что при удалении данных из таблицы может пропасть информация, не связанная напрямую с удаляемыми данными.

Аномалия добавления (anomaly of complement) – проявляется в том, что информацию в таблицу нельзя поместить до тех пор, пока она неполная, либо вставка новой записи требует дополнительного просмотра таблицы.

Applet (applet) – один из видов сетевых приложений, хранимых на Web-серверах в виртуальных каталогах Web-приложений, для загрузки которых используются HTML-страницы с тегом <Applet>.

Атрибут (attribute) – свойство, характеризующее сущность. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок некоторого столбца таблицы.

База данных (database) – поименованная совокупность организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Байт (byte) – основная единица количества информации, включающая восемь битов.

Банк данных (data bank) – разновидность информационной системы, в которой реализованы функции централизованного хранения и обработки информации, организованной в одну или несколько баз данных.

Бит (bit) – минимальная единица количества информации, представляющая один символ двоичного алфавита.

Буфер (buffer) – область оперативной памяти, предназначенная для ускорения обмена между внешней и оперативной памятью.

Буфер обмена (clipboard) – область основной памяти компьютера, с помощью которой организуется обмен данными между приложениями.

Взаимно независимые (mutually independent) атрибуты – такие два или более атрибутов, каждый из которых не является функционально зависимым от других атрибутов.

Внешний ключ (external key) отношения R1 – неключевой атрибут А, значения которого являются значениями ключевого атрибута В другого отношения R2.

Гипермедиа (hypermedia) – гипертекст, содержащий нетекстовые фрагменты (графическое изображение, звук, анимацию).

Гиперссылка (hyperlink) – тип данных, позволяющий хранить в поле таблицы ссылки на файлы или документы, находящиеся вне базы данных.

Гипертекст (hypertext) – текст документа, содержащий ссылки на другие фрагменты текстов произвольных документов, в том числе и этого документа.

Глобальная сеть (wide area network) – сеть, отдельные компоненты которой удалены на значительное расстояние.

Децентрализованная (одноранговая) сеть (peer-to peer network) – локальная вычислительная сеть, функции управления в которой поочередно передаются от одной рабочей станции к другой и которая не имеет выделенных серверов.

Домен отношения (domain of relation) – множество всех возможных значений определенного атрибута отношения.

Домен сети (network domain) – совокупность компьютеров сети, к которым установлены одинаковые правила доступа.

Журнал СУБД (database management system log) – отдельная база данных или часть основной базы данных, непосредственно недоступная пользователю и используемая для записи информации обо всех изменениях базы данных.

Запрос (query) – специальным образом описанное требование, определяющее состав производимых над базой данных операций по выборке или модификации хранимых данных.

Иерархическая модель данных (hierarchical data model) – модель данных, хранящихся в базе, описывающая взаимосвязи с помощью упорядоченного графа (дерева).

Индекс (index) – средство ускорения операции поиска записей в таблице, а также выполнения других операций, использующих поиск: извлечение, модификация, сортировка и т. д.

Индексный файл (index file) – файл, в котором хранится информация индекса.

Инtranет (Intranet) – локальная (корпоративная) информационная сеть, построенная по принципам сети Интернет.

Интернет (Internet) – глобальная всемирная сеть, информация в которой хранится на серверах.

Интерпретатор (interpreter) – программа, которая получает на вход программу на входном языке и по мере распознавания конструкций реализует действия, описываемые этими конструкциями.

Интерпретация (interpretation) – непосредственное исполнение текста исходной программы в ходе просмотра ее текста.

Информационная система (information system) – система, реализующая автоматизированный сбор, обработку и манипулирование данными и включающая технические средства обработки данных, программное обеспечение и обслуживающий персонал.

Информационная система типа клиент-сервер (client-server information system) – система, в которой программы СУБД функционально разделены на две части, называемые сервером и клиентом.

Клиентская программа (front-end program) – программа, отвечающая за интерфейс с пользователем, для чего она преобразует его запросы в команды запросов к серверной части (back-end), а при получении результатов выполняет обратное преобразование и отображение информации для пользователя.

Кластерный индекс (clustered index) – индекс, в котором логический порядок значений ключей совпадает с физическим порядком записей в таблице.

Клиент (client) определенного ресурса в компьютерной сети – компьютер (программа), использующий этот ресурс.

Ключевая таблица (*key table*) – таблица с ключевыми полями.

Ключ отношения, или первичный ключ (*primary key*) – атрибут отношения, однозначно идентифицирующий каждый из его кортежей.

Компиляция (*compilation*) – преобразование исходного текста программы в последовательность исполняемых машинных команд.

Компилятор (*compiler*) – транслятор с языка программирования высокого уровня.

Компьютер-клиент (*computer-client*) – ЭВМ сети, обращающаяся за ресурсами к компьютерам-серверам.

Компьютер-сервер (*computer-server*) – ЭВМ сети, предоставляющая свои ресурсы другим компьютерам сети.

Курсор (*cursor*) – своеобразный указатель, используемый для перемещения по наборам записей при их обработке.

Логическая целостность (*logical integrity*) – отсутствие логических ошибок в базе данных, к которым относятся нарушение структуры базы данных или ее объектов, удаление или изменение связей между объектами и т. д.

Локальная вычислительная сеть (*local area network*) – сеть, в которой компьютеры расположены на расстоянии до нескольких километров и обычно соединены при помощи скоростных линий связи.

Макрос (*macro*) – последовательность макрокоманд встроенного языка программирования СУБД, автоматизирующих выполнение последовательности действий пользователя.

Многомерная модель данных (*multilevel data model*) – модель данных, обеспечивающая многомерное логическое представление структуры информации при ее описании и в операциях манипулирования данными.

Модель представления данных (*data model*) – логическая структура данных, хранящихся в базе данных.

Мультимедиа (*multimedia*) – данные различной природы: звуковые, видео-, графические, текстовые, с различными эффектами отображения на экране. В широком смысле мультимедиа означает совокупность технологий производства и применения аппаратных и программных средств, позволяющих поддерживать работу с перечисленными видами информации.

Мэйнфрейм (*mainframe*) – многопользовательская централизованная вычислительная система.

Нормальная форма (*normal form*) – форма задания ограничения типа функциональных зависимостей для устранения аномалий при выполнении операций над отношениями базы данных.

Обозреватель (*browser*) – программа-навигатор, с помощью которой производится доступ пользователей к ресурсам сети Интернет.

Объектно-ориентированная модель данных (*object-oriented data model*) – модель данных, которая позволяет между записями базы данных и функциями их обработки устанавливать взаимосвязи с помощью механизмов, подобных соответствующим средствам в объектно-ориентированных языках программирования.

Ограничения целостности (*integrity constraints*) – условия, которым должны удовлетворять хранимые в базе данные.

Отношение (*relation*) – множество, представляющее двумерной таблицей, состо-

ящей из строк и столбцов данных.

Отчет (report) – объект базы данных, основное назначение которого – описание и вывод на печать документов на основе данных базы.

Переменные окружения (environment variables) – системные переменные, предназначенные для хранения служебной информации

Первичный ключ, или ключ отношения, или ключевой атрибут (primary key) – атрибут отношения, однозначно идентифицирующий каждый из его кортежей.

Поле связи, или ключ связи (connection field) – поле, с помощью которого производится логическое связывание таблиц. Поле связи, подобно ключу таблицы, состоит из одного или нескольких полей.

Постреляционная модель данных (post-relational data model) – расширенная реляционная модель, снимающая ограничение неделимости данных, хранящихся в записях таблиц.

Приложение, или прикладная программа (application) – программа или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной задачи. Приложения, разработанные в среде СУБД, часто называют приложениями СУБД, а приложения, разработанные вне СУБД, – внешними приложениями.

Приложение базы данных (database application) – программа или комплекс программ, использующих базу данных и обеспечивающих автоматизацию обработки информации из некоторой предметной области.

Приложение Windows (Windows application) – программа, выполнимая под управлением Windows.

Псевдокомпиляция (pseudocompilation) – промежуточный вариант между компиляцией и интерпретацией, при котором исходная программа путем компиляции преобразуется в промежуточный код (псевдокод) и записывается на диск.

Рабочая группа (workgroup) – группа пользователей, для которых определена единая технология работы.

Рабочая станция (workstation) – как правило, персональная ЭВМ, являющаяся рабочим местом пользователя сети.

Реляционная алгебра (relational algebra) – теоретический (процедурный) язык запросов.

Реляционное исчисление (relational calculus) – теоретический (непроцедурный, описательный, декларативный) язык запросов.

Репликация (replication) – создание специальных копий (реплик) базы данных, с которыми пользователи могут работать одновременно на разных рабочих станциях.

Реляционная модель данных (relational data model) – модель данных, хранящихся в базе, описывающая взаимосвязи элементов данных в виде отношения (таблицы).

Сервер базы данных (database server) – программа, выполняющая функции управления и защиты базы данных. В случаях, когда вызов функций сервера выполняется на языке SQL, его называют SQL-сервером.

Сервер (server) определенного ресурса в компьютерной сети – компьютер (программа), управляющий этим ресурсом.

Сервлет (servlet) – вид сетевого приложений, разрабатываемый с помощью языка Java, который служит для расширения возможностей Web-серверов, аналогично тому как апплеты расширяют возможности обозревателя (клиента).

Сетевая модель данных (network data model) – модель данных, хранящихся в базе, описывает взаимосвязи элементов в виде графа произвольного вида (сети).

Сетевая СУБД (network database management system) – система управления базами данных с произвольной моделью данных, ориентированная на использование в сети.

Сеть (network) – совокупность компьютеров, объединенных средствами передачи данных.

Система управления базами данных (database management system) – комплекс языковых и программных средств, предназначенный для создания, ведения и совместного использования баз данных.

Словарь данных (data dictionary) – подсистема банка данных, предназначенная для централизованного хранения информации о структурах данных, взаимосвязях файлов базы данных друг с другом, типах данных и форматах их представления, принадлежности пользователям, кодах защиты и разграничения доступа и т. п.

Сущность (essence) – объект любой природы, данные о котором хранятся в базе данных.

Схема отношения (scheme of relation) – список имен атрибутов отношения.

Сценарий (script) – программа на макроязыке сценариев (например, JScript или VBScript), помещаемая HTML-документ и выполняемая в режиме интерпретации.

Таблица (table) – основная единица хранения данных в базе.

Тег (tag) – специальный управляющий элемент, используемый в HTML-документах, несущий в себе служебную информацию для обозревателя и позволяющий задавать различные режимы форматирования.

Технология клиент-сервер (client-server) – технология, при которой процесс обработки информации распределен между клиентом и сервером.

Транзакция (transaction) – последовательность операций над базой данных, отслеживаемая системой управления базами данных от начала до завершения как единое целое.

Транзитивная зависимость (transitive dependence) атрибута С от атрибута А – такая зависимость атрибутов, при которой имеет место функциональная зависимость атрибута В от атрибута А и функциональная зависимость атрибута С от атрибута В.

Транслятор (translator) – программа, получающая на входе исходную программу и порождающая на выходе функционально эквивалентную исходной объектную программу.

Триггер (trigger) – разновидность хранимой процедуры, которая автоматически вызывается при возникновении определенных событий в базе данных.

Тупик (deadlock) – ситуация, при которой потребителям ресурсов невозможно использовать их совместно.

Файл (File) – логически связанная совокупность данных (программ, текстов, изображений и др.) определенной длины, имеющая имя.

Файл-сервер (file-server) – компьютер, предназначенный для организации управления файлами в сети.

Форма (form) – объект базы данных, в котором разработчик размещает элементы управления, служащие для ввода, отображения и изменения данных в полях.

Форма HTML (HTML form) – основное средство организации интерактивного взаимодействия в Интернете при разработке Web-приложений, служащее для пересылки данных от пользователя к Web-серверу.

Функциональная зависимость (functional dependence) атрибута В от атрибута А – такая зависимость атрибутов, при которой каждому значению атрибута А соответствует одно значение атрибута В.

Хранимая процедура (storage procedure) – программа (процедура) обработки данных, хранящаяся и выполняемая на компьютере-сервере.

Хранимые команды (storage commands) – поименованные совокупности команд, получаемые в результате компиляции SQL-запросов.

Целостность (integrity) – свойство базы данных, означающее, что она содержит полную, непротиворечивую и адекватно отражающую предметную область информацию.

Частичная зависимость, или частичная функциональная зависимость (partial functional dependence) атрибутов – зависимость неключевого атрибута от части составного ключа.

Электронная почта (E-mail) – способ доступа в сети Internet, позволяющий пересылать небольшие файлы любых типов (тексты, изображения, звук) по адресам электронной почты в любую точку планеты за короткий промежуток времени.

Язык манипулирования данными (data manipulation language) – совокупность конструкций, обеспечивающих выполнение основных операций по работе с данными: ввод, модификацию и выборку данных по запросам.

Язык определения данных (data definition language) – высокоуровневый язык декларативного типа, предназначенный для описания логической структуры данных.

API (Application Programming Interface – интерфейс прикладного программирования) – программные средства организации взаимодействия между пользователем и приложением.

ASP (Active Server Page – активная серверная страница) – документ, включающий HTML-шаблон и использующий серверный сценарий на языке JScript или VBScript и запросы к БД на языке SQL для динамического формирования HTML-страниц.

BDE (Borland Database Engine – процессор баз данных) – стандартизованное средство доступа к базам данных, разработанное фирмой Borland.

CASE-средство – программное средство, поддерживающее процессы создания и/или сопровождения информационных систем.

CASE-система – набор CASE-средств, имеющих определенное функциональное предназначение и выполненных в рамках единого программного продукта.

CASE-технология – методология проектирования информационных систем плюс инструментальные средства, позволяющие наглядно моделировать предметную область, анализировать ее модель на всех этапах разработки и сопровождения информационной системы и разрабатывать приложения.

CGI (Common Gateway Interface - процессор баз данных) – стандартный протокол взаимодействия между Web-сервером и модулями расширения, которые могут применяться для выполнения дополнительных функций, не поддерживаемых сервером. Служит для создания модулей расширения сервера.

CORBA (Common Object Request Broker Architecture – архитектура брокера общих объектных запросов) – стандарт технологии для информационных систем с распределенной обработкой.

CSS (Cascading Style Sheets – каскадные таблицы стилей) – язык, предназначенный для описания внешнего вида HTML-документов.

IDAPI (Integrated Database Application Program Interface) – стандартный интерфейс доступа к базам данных, разработанный фирмой Borland, включающий драйверы баз данных распространенных форматов и утилиты настройки драйверов и псевдонимов.

ISAPI/NSAPI (Internet Server API/Netscape Server API) – интерфейсы прикладного программирования, разработанные фирмами Microsoft и Netscape соответственно. Предназначены для разработки дополнительных модулей расширения Web-сервера.

FTP (File Transfer Protocol – протокол передачи файлов) – протокол передачи файлов любых типов в сети.

Gopher – протокол работы с информационными ресурсами Internet с помощью команд в виде системы меню.

HTML (HyperText Markup Language – язык разметки гипертекста) – стандартизованный язык описания, позволяющий создавать Web-документы (HTML-документы) для сети Интернет.

HTTP (HyperText Transfer Protocol – протокол передачи гипертекста) – один из самых распространенных протоколов, предназначен для передачи данных различных форматов между обозревателем и сервером.

ODBC (Open Database Connectivity) – интерфейс прикладного программирования, разработанный фирмой Microsoft, в виде библиотеки функций,ываемых из различных программных сред и позволяющих приложениям унифицированно обращаться на SQL к базам данных различных форматов.

OLE (Object Linking and Embedding – связывание и встраивание объектов) – один из наиболее распространенных стандартов интеграции, положенный в основу разработки программных систем.

OLE DB (Object Linking and Embedding Database – связывание и встраивание объектов баз данных) – стандартный интерфейс, представляющий собой универсальную технологию доступа к любым источникам данных через интерфейс COM.

OSI (Open Systems Interconnection reference model) – эталонная модель взаимодействия открытых систем.

PHP (Personal Home Page tools – средства персональных домашних страниц) – язык обработки сценариев, применяемый для разработки Web-приложений.

QBE (Query By Example) – язык запросов по образцу, позволяющий подготавливать запросы в наглядной форме.

SQL (Structured Query Language) – структурированный язык запросов, представляющий собой стандартизованное средство описания запросов к базам данных.

URL (Uniform Resource Locator) – унифицированный указатель ресурсов, используется для адресования Web страниц и других ресурсов Интернет.

TCP/IP (Transmission Control Protocol/Internet Protocol – протокол управления передачей данных/Протокол Интернет) – протокол, используемый для передачи данных в сети Internet и во многих глобальных и локальных сетях.

XML (eXtensible Markup Language – расширяемый язык разметки) – развитие языка HTML, с его помощью выполняется наполнение создаваемого документа содержанием с указанием разметки, а также определяются структура документа и типы хранимых в нем данных.

WWW (World Wide Web – всемирная паутина) – средство организации сетевых ресурсов, построенное на основе гипертекстового представления информации.

Содержание

Предисловие	5
-------------------	---

Часть 1. Основы построения баз данных

1. Введение в базы данных	9
1.1. Базы данных и информационные системы	9
1.2. Архитектура информационной системы	12
1.3. Системы управления базами данных	14
1.4. Локальные информационные системы	20
1.5. Способы разработки и выполнения приложений	22
1.6. Схема обмена данными при работе с БД	25
Контрольные вопросы и задания	27
Литература	28
2. Модели и типы данных	29
2.1. Иерархическая модель	29
2.2. Статевая модель	32
2.3. Реляционная модель	33
2.4. Постреляционная модель	34
2.5. Многомерная модель	37
2.6. Объектно-ориентированная модель	41
2.7. Типы данных	44
Контрольные вопросы и задания	47
Литература	48
3. Реляционная модель данных	49
3.1. Определение реляционной модели	49
3.2. Индексирование	53
3.3. Связывание таблиц	57
3.4. Контроль целостности связей	63
3.5. Теоретические языки запросов	65
3.6. Реляционная алгебра	68
3.7. Реляционное исчисление	79
3.8. Язык запросов по образцу QBE	84
3.9. Структурированный язык запросов SQL	98
Контрольные вопросы и задания	107
Литература	108
4. Информационные системы в сетях	109
4.1. Основные понятия	109
4.2. Модели архитектуры клиент-сервер	114
4.3. Управление распределенными данными	123
4.4. Информационные системы в локальных сетях	133
4.5. Информационные системы в Интернете и интранете	139
Контрольные вопросы и задания	147
Литература	148

Участие авторов в подготовке материалов:

Хомоненко А. Д., д.т.н., профессор – предисловие, разделы 1, 5, 7, 11–18, подразделы 3.1, 9.3, 9.4

Цыганков В. М. к.т.н. – разделы 1–5, 8–10, 13, подразделы 7.1, 7.5, 7.6 и 7.8, приложения

Мальцев М. Г. к.т.н., доцент – разделы 6 и 12, подразделы 1.2, 3.1, 5.1, 5.2 и 5.4

Мещеряков Е. В. к.т.н. – разделы 14–18

Гридин В. В. – подразделы 10.1–10.4

Кирюшкин С. В. – подраздел 9.4

Часть 2. Проектирование и использование баз данных	
5. Проектирование баз данных	149
5.1. Проблемы проектирования	149
5.2. Метод нормальных форм	154
5.3. Рекомендации по разработке структур	171
5.4. Следование целостности	172
Контрольные вопросы и задания	174
Литература	174
6. Метод сущность-связь	175
6.1. Основные понятия метода	175
6.2. Этапы проектирования	180
6.3. Правила формирования отношений	180
6.4. Пример проектирования БД учебной части	190
Контрольные вопросы и задания	193
Литература	194
7. Средства автоматизации проектирования	195
7.1. Основные определения	195
7.2. Модели жизненного цикла	198
7.3. Модели структурного проектирования	199
7.4. Объектно-ориентированные модели	203
7.5. Классификация CASE-средств	208
7.6. Системы структурного типа	210
7.7. Объектно-ориентированные системы	216
7.8. Рекомендации по применению CASE-систем	218
Контрольные вопросы и задания	219
Литература	220
8. Использование баз данных	221
8.1. Настройка и администрирование	221
8.2. Защита информации	224
Контрольные вопросы и задания	240
Литература	241
9. Дополнительные вопросы применения баз данных	242
9.1. Программно-аппаратные платформы	242
9.2. Перспективы развития СУБД	259
9.3. Стандартизация баз данных	262
9.4. Характеристика технологии ADO.NET	270
Контрольные вопросы и задания	274
Литература	275
Часть 3. Современные СУБД и их применение	
10. СУБД Access 2002	277
10.1 Общая характеристика	277
10.2. Новые возможности Microsoft Access 2002	282
10.3. Средства поддержки проектирования	283
10.4. Создание основных элементов БД	286
10.5. Работа с гиперссылками	308
10.6 Использование языка SQL	312
10.7 Защита баз данных	321
10.8 Скрытие объектов баз данных	337
10.9 Обслуживание баз данных	338
10.10. Репликация баз данных	340
10.11. Работа с мультимедиа-данными	347
10.12. Создание файлов приложений	351

10.13.Страницы доступа к данным	352
10.14. Разработка проекта	357
Контрольные вопросы и задания	362
Варианты индивидуального задания	363
Литература	365
11. Borland C++ Builder	366
11.1. Пользовательский интерфейс	366
11.2. Характеристика проекта	371
11.3. Компиляция и выполнение проекта	381
11.4. Разработка приложения	382
11.5. Средства интегрированной среды разработки	394
11.6. Базы данных и средства работы с ними	397
11.7. Создание таблиц базы данных	413
11.8. Создание приложения BDE	420
11.9. Работа с отчетами	422
Контрольные вопросы и задания	431
Литература	433
12. СУБД Visual FoxPro 8.0	434
12.1. Общая характеристика	434
12.2. Новые возможности Visual FoxPro 8.0	435
12.3. Элементы проекта	436
12.4. Интерфейс Visual FoxPro	439
12.5. Средства автоматизации разработки	440
12.6. Создание баз данных	441
12.7. Таблицы и индексы	448
12.8. Организация межтабличных связей	455
12.9. Обеспечение ссылочной целостности	457
12.10. Создание запросов	461
Контрольные вопросы и задания	474
Литература	475
13. Microsoft SQL Server 2000	476
13.1. Характеристика SQL Server	476
13.2. Язык запросов Transact-SQL	487
13.3. Системные базы данных и таблицы	488
13.4. Создание баз данных	491
13.5. Работа с таблицами	493
13.6. Индексы и ключи	501
13.7. Хранимые процедуры и триггеры	504
13.8. Обеспечение безопасности	509
13.9. Организация взаимодействия клиент-сервер	514
13.10. Обработка данных с помощью ODBC	519
Контрольные вопросы и задания	523
Литература	524
Часть 4. Публикация баз данных в Интернете	
14. Введение в технологии публикации	525
14.1. Основы Интернет-технологий	526
14.2. Состав и теги HTML-документа	537
14.3. Особенности XML-документа	553
Контрольные вопросы и задания	565
Литература	566
15. Web-приложения и Web-серверы	567
15.1. Принципы функционирования Web-приложений	567

15.2. Архитектура Web-приложений, публикующих БД	578
15.3. Обзор Web-серверов	590
15.4. Использование Personal Web-server	598
15.5. Использование Microsoft Internet Information Server	602
15.6. Использование Apache для Microsoft Windows 9X/2000	609
15.7. Варианты создания Web-узла	614
Контрольные вопросы и задания	615
Литература	616
16. Интерфейсы программирования Web-приложений	617
16.1. Общий интерфейс взаимодействия CGI	617
16.2. Интерфейс программирования серверных приложений ISAPI	644
Контрольные вопросы и задания	659
Литература	660
17. Публикация БД с использованием XML	661
17.1. XML как средство обмена данными	661
17.2. Создание и обработка XML-документов	662
17.3. Сценарий для отображения XML-документа	664
17.4. Формирование XML-документа на основе базы данных	669
17.5. Размещение данных из XML-документа в базе данных	673
Контрольные вопросы и задания	679
Литература	679
18. Публикация БД средствами Microsoft Access	681
18.1. Характеристика вариантов публикации	681
18.2. Страницы доступа к данным	682
18.3. Серверные страницы	693
18.4. Статические файлы HTML	710
Контрольные вопросы и задания	714
Литература	714
Приложения	
Приложение 1. Аксиомы вывода функциональных зависимостей	715
Приложение 2. Краткое описание языка SQL в MS Access	718
Перечень терминов	726